Flexible und leistungsfähige Nutzung Dienst-spezifischer Netze auf Endsystemen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)
genehmigte

Dissertation

von

Hans Wippel
aus Bruchsal

Tag der mündlichen Prüfung:	8. Juli 2015
Erste Gutachterin:	Prof. Dr. Martina Zitterbart
Zweiter Gutachter:	Prof. Dr. Wolfgang Kellerer

DOI: 10.5445/IR/1000048868

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die über das Internet bereitgestellten Dienste haben sich über die letzten Jahrzehnte stark weiterentwickelt. Wurden in den Anfangszeiten nur Dienste zum Austausch von Dateien oder Nachrichten verwendet, so gibt es heute eine Vielzahl von unterschiedlichen Diensten. Heutzutage sind beispielsweise Dienste wie Video-Streaming, Online-Gaming oder Online-Banking sehr weit verbreitet. Darüber hinaus unterliegt das Internet einem stetigen Wandel. Stets werden neue Dienste über das Internet betrieben und es ist nicht abzusehen, welche in Zukunft noch angeboten werden. Schon heute stellen die im Internet angebotenen Dienste unterschiedlichste, teils gegensätzliche Anforderungen an die Kommunikationsprotokolle im Internet. Mögliche Anforderungen sind z.B. ein hoher Durchsatz, hohe Zuverlässigkeit, hohe Sicherheit, die Bewahrung der Privatsphäre der Nutzer, oder ein niedriger Energiebedarf. Es gibt kein einzelnes Kommunikationsprotokoll, das allen Anforderungen beliebiger Dienste gerecht wird. Ebenso stellen die unterschiedlichen Anforderungen die im Internet vorhandenen Protokolle bereits vor große Herausforderungen. Daher wurden in Forschungsarbeiten zahlreiche Protokolle entwickelt, die an spezielle Anforderungen angepasst sind. Neue Protokolle im heutigen Internet einzuführen, ist allerdings ein schwieriger und langwieriger Prozess. Selbst Protokolle, die bereits vor Jahrzehnten erfolgreich lange Test- und Standardisierungsprozesse durchlaufen haben, sind heutzutage im Internet kaum verfügbar, was Beispiele wie IPv6 [21], SCTP [106] und DCCP [60] verdeutlichen. Aus diesem Grund wurden Dienst-spezifische Netze (DsN) entwickelt. Diese sind virtuelle Netze, die an die Anforderungen individueller Dienste angepasst sind. Innerhalb eines DsN werden hierfür Dienst-spezifische Protokolle verwendet, die an die Eigenschaften und Anforderungen des erbrachten Dienstes angepasst sind. Dank Virtualisierungstechniken können DsNs über der existierenden Netz-Infrastruktur schnell ausgebracht werden und in ihnen können beliebige Protokolle verwendet werden. Eine Interoperabilität zwischen Dienst-spezifischen Protokollen unterschiedlicher DsNs ist nicht nötig. Mit Dienst-spezifischen

Netzen ist es also möglich, anstatt eines Netzes für alle Dienste, wie es im heutigen Internet üblich ist, individuelle, virtuelle Netze für verschiedene Dienste zu verwenden. Dabei wird jeweils ein Dienst in einem Dienst-spezifischen Netz erbracht und die Kommunikationsprotokolle des Dienst-spezifischen Netzes sind an die Anforderungen des Dienstes angepasst. Dienst-spezifische Netze bieten dadurch ein hohes Maß an Flexibilität und fördern Innovation sowohl innerhalb der DsNs als auch in der darunter liegenden Netz-Infrastruktur. Durch die Entkoppelung können sich Dienst-spezifische Netze und die Netz-Infrastruktur unabhängig voneinander weiterentwickeln. In den Dienst-spezifischen Netzen können beliebige Protokolle verwendet werden. In der Netz-Infrastruktur können neue Protokolle ausgebracht werden, ohne die Dienste oder Dienst-spezifischen Protokolle in den Dienst-spezifischen Netzen anpassen zu müssen.

Allerdings existierten bisher noch Probleme, die gelöst werden mussten, um Dienst-spezifische Netze tatsächlich nutzen zu können. Eine offene Fragestellung war, wie Endsysteme auf Dienst-spezifische Netze zugreifen und diese effizient nutzen können. Besondere Herausforderungen stellten dabei die Heterogenität und Dynamik Dienst-spezifischer Netze dar.

Dienst-spezifische Netze und ihre Herausforderungen

Ein Dienst-spezifisches Netz ist ein virtuelles Netz, dessen Protokolle an die Anforderungen eines individuellen Dienstes angepasst sind. Ein Dienste-Anbieter legt für die Realisierung eines Dienstes in einem Dienst-spezifischen Netz die Topologie des DsN und die Protokolle, die innerhalb des DsN verwendet werden, fest. Das Dienst-spezifische Netz wird dann in der Netz-Infrastruktur ausgebracht. Greifen Nutzer von Endsystemen auf Dienste, die in Dienst-spezifischen Netzen realisiert werden, zu, ergeben sich folgende Herausforderungen, die anhand von Abbildung 1.1 erläutert werden.

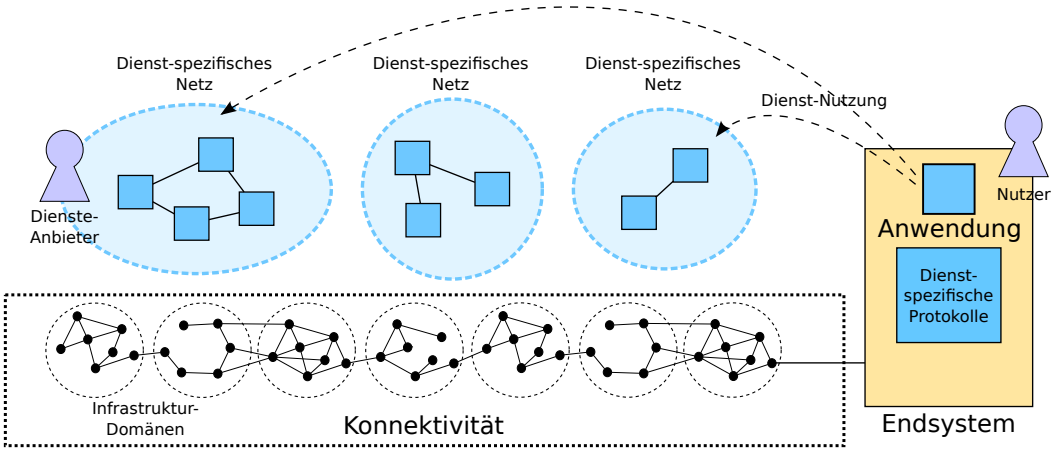

Abbildung 1.1: *Dienst-spezifische Netze und Nutzung durch Endsysteme*

In der Abbildung werden drei Dienst-spezifische Netze dargestellt, die aus virtuellen Knoten, beispielsweise Virtuelle Maschinen, und virtuellen Verbindungen zwischen ihnen bestehen. Jedes Dienst-spezifische Netz stellt einen Dienst bereit. Stellvertretend für alle Dienst-spezifischen Netze ist bei einem ein Dienste-Anbieter eingezeichnet. Dienste-Anbieter betreiben Dienst-spezifische Netze, um über sie Dienste anzubieten. Die Dienst-spezifischen Netze werden über die Netzwerk-Infrastruktur betrieben, die aus verschiedenen Infrastruktur-Domänen besteht und Konnektivität zwischen Dienst-spezifischen Netzen und Endsystemen ermöglicht. Beispielhaft ist in der Abbildung ein Endsystem eingezeichnet, das mit einer der Infrastruktur-Domänen verbunden ist. Auf dem Endsystem verwendet ein Nutzer eine Anwendung, die auf die Dienste in zwei Dienst-spezifischen Netzen zugreift. Für den Zugriff auf die Dienste betreibt das Endsystem die Dienst-spezifischen Protokolle der entsprechenden Dienst-spezifischen Netze.

(1) *Zugriff von Endsystemen auf Dienst-spezifische Netze*: Eine wichtige Fragestellung ist, wie Endsysteme auf Dienst-spezifische Netze zugreifen können und wie der Datenaustausch zwischen Endsystemen und Dienst-spezifischen Netzen effizient realisiert werden kann. Um auf einen Dienst zugreifen zu können, muss ein Endsystem wie in Abbildung 1.1 zunächst ein Dienst-spezifisches Netz finden, das den Dienst anbietet. Ist ein solches gefunden, benötigt das Endsystem die für die Nutzung des Dienst-spezifischen Netzes benötigten Protokolle. Außerdem muss sich das Endsystem zu dem Dienst-spezifischen Netz verbinden, indem es eine Virtuelle Verbindung über die vorhandene Netz-Infrastruktur zu dem Dienst-spezifischen Netz aufbaut. Ein Endsystem hat allerdings kein Vorwissen darüber, welche Dienste ein Nutzer während der Laufzeit verwenden wird. Daher müssen diese Probleme zur Laufzeit, wenn der Nutzer auf einen Dienst zugreifen will, gelöst werden. Außerdem sollte der Zugriff auf Dienst-spezifische Netze sowohl über das heutige Internet als auch über zukünftige Kommunikationsnetze funktionieren.

(2) *Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen eines Endsystems*: Unterhält ein Endsystem Verbindungen zu verschiedenen Dienst-spezifischen Netzen, betreibt es dementsprechend mehrere Virtuelle Verbindungen und Dienst-spezifische Protokolle. Virtuelle Verbindungen und Dienst-spezifische Protokolle verbrauchen Netz-, Speicher- und Prozessor-Ressourcen des Endsystems und können sich durch ihren Ressourcen-Verbrauch gegenseitig beeinflussen. Wenn beispielsweise ein Protokoll eines Dienst-spezifischen Netzes aufwändige Berechnungen auf den übertragenen Daten ausführt, könnte hierdurch die Verarbeitung eines zeitkritischen Protokolls eines anderen Dienst-spezifischen Netzes verzögert werden. Zum Einhalten der Anforderungen Dienst-spezifischer Netze ist es also nötig, den Ressourcen-Verbrauch dieser Netze zu regulieren. Eine besondere Herausforderung bei der Verwaltung von Ressourcen ist die Heterogenität Dienst-spezifischer Netze und die Dynamik durch neu hinzukommende und wegfallende Verbindungen zu DsNs. Ein Endsystem besitzt kein Vorwissen über die

Verbindungen, die zur Laufzeit zu Dienst-spezifischen Netzen aufgebaut werden, und kein Wissen über die Protokolle, die in den DsNs verwendet werden.

(3) *Schneller Datenaustausch zwischen Anwendungen auf einem Endsystem und Dienst-spezifischen Netzen*: Bei der Verwendung Dienst-spezifischer Netze sollten keine Leistungsengpässe auf einem Endsystem entstehen und es sollte eine möglichst hohe Leistung erreicht werden können. Zum Beispiel sollte ein auf hohe Datenraten optimiertes Protokoll auch eine hohe Datenrate erreichen können. Hierfür muss das Endsystem einen schnellen Datenaustausch zwischen Anwendungen und dem Netz ermöglichen. Eine besondere Herausforderung ist, dass die Eigenschaften und somit auch die Paketlängen zukünftiger Dienst-spezifischer Netze nicht absehbar sind. Nutzer sollten daher hohe Datenraten bei beliebigen Paketlängen erhalten. Allerdings stellen gerade kleine Pakete ein Problem dar, da mit ihnen hohe Paketraten möglich sind und dadurch die zur Verarbeitung eines Pakets zur Verfügung stehende Zeit sehr gering ist.

1.1 Zielsetzung und Beiträge

Diese Arbeit hat sich zum Ziel gesetzt, eine flexible und leistungsfähige Nutzung Dienst-spezifischer Netze auf Endsystemen zu ermöglichen. Hierzu wurden Lösungen für die folgenden Herausforderungen erarbeitet: (1) Zugriff von Endsystemen auf Dienst-spezifische Netze, (2) Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen eines Endsystems, (3) Schneller Datenaustausch zwischen Anwendungen auf einem Endsystem und Dienst-spezifischen Netzen. Diese Arbeit umfasst somit die folgenden Beiträge.

(1) *Flexibles Zugriffsverfahren für Dienst-spezifische Netze*: Ein flexibles Zugriffsverfahren für Dienst-spezifische Netze wurde entwickelt, das Endsystemen ermöglicht, Dienst-spezifische Netze für Dienste zu finden, die benötigten Protokolle zu beziehen und Verbindungen zu Dienst-spezifischen Netzen aufzubauen, wenn der Nutzer auf Dienste zugreift. Das in dieser Arbeit vorgestellte Verfahren löst diese Probleme transparent für den Nutzer. Der Ansatz verwendet einen Mapping-Dienst zum Ablegen von Informationen, einen Speicher-Dienst zum Speichern von Protokollen und ein Basisnetz für die Konnektivität zwischen Endsystemen, dem Mapping-Dienst und dem Speicher-Dienst. Greift der Nutzer auf einen Dienst zu, nutzt das Endsystem den Mapping-Dienst, um den vom Nutzer angeforderten Dienst in ein Dienst-spezifisches Netz, eine Liste von Zugangspunkten zum Dienst-spezifischen Netz und eine Liste von Protokollen aufzulösen. Anschließend bezieht das Endsystem die Protokolle aus dem Speicher-Dienst und baut eine Verbindung zum Dienst-spezifischen Netz auf. Durch den modularen Aufbau der Lösung wird ein hoher Grad an Flexibilität erreicht. Es ist möglich, sowohl die Methoden, die für das Beziehen der Dienst-spezifischen Protokolle eingesetzt werden, als auch die Methoden, die für das Aufbauen der Verbindungen zu Dienst-spezifischen Netzen

verwendet werden, zur Laufzeit auszutauschen. Nur die Verbindung zum Basisnetz und die darin verwendeten Protokolle müssen zur Konfigurationszeit vorgegeben werden. Durch die Flexibilität kann der Ansatz zukünftige Entwicklungen im Internet bzw. dem Kommunikationsnetz, wie z.B. neue Kommunikationsprotokolle, unterstützen.

(2) *Flexible Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen*: Ein Hierarchisches Verwaltungssystem wurde entworfen, das die flexible Verwaltung der für Dienst-spezifische Netze benötigten Netz-, Speicher-, und Prozessor-Ressourcen auf einem Endsystem erlaubt. Das Verwaltungssystem erlaubt, die auf einem Endsystem verfügbaren Ressourcen auf verschiedene Dienst-spezifische Netze und die darin verwendeten Protokolle aufzuteilen und dabei Nutzer-Richtlinien zu beachten. Das Verwaltungssystem besteht aus mehreren Verwaltungskomponenten, die in einer baumförmigen Hierarchie organisiert sind. Dabei bildet eine für das Endsystem zentrale Verwaltungskomponente die Wurzel des Baumes. Auf der nächsten Baum-Ebene befindet sich für jede Verbindung zu einem Dienst-spezifischen Netz eine Verwaltungskomponente, die an das Dienst-spezifische Netz angepasst ist. Auf der Blatt-Ebene des Baumes befindet sich schließlich für jedes Protokoll eine Verwaltungskomponente, die an das entsprechende Protokoll angepasst ist. Entlang der Hierarchie können sich die Verwaltungskomponenten Ressourcen zuweisen oder entziehen. Auf die Veränderungen der ihnen zugewiesenen Ressourcen können die Verwaltungskomponenten individuell reagieren. Dank des modularen Aufbaus und der Hierarchie können neue Verwaltungskomponenten hinzugefügt und bestehende entfernt werden, ohne das gesamte Verwaltungssystem anpassen zu müssen. Detailliertes Wissen über die Verwaltung der Netze oder Protokolle wird nur in den entsprechenden Verwaltungskomponenten benötigt.

(3) *Flexibles und leistungsfähiges Rahmenwerk für Dienst-spezifische Netze auf Endsystemen*: Ein Rahmenwerk für Dienst-spezifische Netze auf Endsystemen wurde entwickelt, das die Flexibilität Dienst-spezifischer Netze mit dem Erreichen hoher Datenraten selbst bei kleinen Paketen kombiniert. Das Rahmenwerk setzt verschiedene Optimierungen ein, um eine hohe Leistung zu erreichen. Beispielsweise greift das Rahmenwerk direkt auf die Netzwerkschnittstellen des Endsystems zu. Nachrichten von Anwendungen können über Dienst-spezifische Netze versendet oder empfangen werden, ohne die entsprechenden Daten kopieren zu müssen. Darüber hinaus werden Prozessor-Kerne explizit der Paketverarbeitung zugeordnet. Außerdem wird eine angepasste Hash-Tabelle eingesetzt, die es erlaubt, eintreffende Pakete ohne vorheriges Wissen über die genauen Paketformate und -Inhalte effizient den Verbindungen zu Dienst-spezifischen Netzen zuzuordnen. Es konnte gezeigt werden, dass das Rahmenwerk in der Lage ist, eine mehr als acht mal höhere Leistung als eine auf der Socket-API basierende Lösung zu erreichen und eine 10 Gigabit/s Netzwerkschnittstelle bereits mit kleinen Paketlängen auszulasten.

1.2 Zugrunde liegende Veröffentlichungen

Die Beiträge dieser Arbeit wurden in nationalen und internationalen Konferenzen veröffentlicht. Das flexible Zugriffsverfahren auf Dienst-spezifische Netze für Endsysteme wurde in den folgenden Veröffentlichungen vorgestellt.

- Das Zugriffsverfahren wurde präsentiert in: H. Wippel und O. Hanka. “End User Node Access to Application-Tailored Future Networks”. In: *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. 2012, S. 1–7. DOI: 10.1109/ICCCN.2012.6289290.

- Sicherheitsaspekte des Zugriffsverfahrens wurden vorgestellt in: O. Hanka und H. Wippel. “Secure Deployment of Application-tailored Protocols in Future Networks”. In: *Network of the Future (NOF), 2011 International Conference on the*. 2011, S. 10–14. DOI: 10.1109/NOF.2011.6126668.

- Ein Demonstrator des Zugriffsverfahrens wurde gezeigt in: H. Wippel und O. Hanka. “Deployment of Application-Tailored Protocols in Future Networks”. In: *Proceedings of the 11th Euroview Future Internet Workshop*. Würzburg, Germany, 2011.

Die Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen auf einem Endsystem wurde in den folgenden Veröffentlichungen präsentiert.

- Das Hierarchische Verwaltungssystem wurde vorgestellt in: H. Wippel u. a. “Hierarchical Node Management in the Future Internet”. In: *Communications Workshops (ICC), 2011 IEEE International Conference on*. 2011, S. 1–5. DOI: 10.1109/iccw.2011.5963590.

- Die Idee des Hierarchischen Verwaltungssystems wurde vorgestellt in: H. Wippel, T. Gamer und D. Martin. “A Hierarchical Node Management System for Application-tailored Network Protocols and Architectures”. In: *5th GI/ITG KuVS Workshop on Future Internet*. Gesellschaft für Informatik, Juni 2010.

- Eine Verwendung des Hierarchischen Verwaltungssystems zur Angriffserkennung wurde als Poster gezeigt in: T. Gamer und H. Wippel. “A Collaborative Attack Detection and its Challenges in the Future Internet”. In: *Proc. of the Joint ITG, ITC, and Euro-NF Workshop "Visions of Future Generation Networks"(EuroView)*. Aug. 2010.

Das flexible und leistungsfähige Rahmenwerk für Dienst-spezifische Netze auf Endsystemen wurde in der folgenden Veröffentlichung präsentiert.

- DPDK-NENA, das leistungsfähige Rahmenwerk für Dienst-spezifische Netze auf Endsystemen wurde vorgestellt in: H. Wippel. “DPDK-based Implementation of Application-tailored Networks on End User Nodes”. In: *Network of the Future (NOF), 2014 International Conference on the*. 2014.

Darüber hinaus wurden im Rahmen dieser Arbeit Lösungen und Konzepte für NENA [74] und Dienst-spezifischen Netze entwickelt, die im Rahmen verschiedener Veröffentlichungen, Demonstratoren, Poster, Vorträge und Technischer Berichte veröffentlicht worden sind. Der Entwurf und die Verwendung von Dienst-spezifischen Protokollen wurde in [115, 6, 96] vorgestellt und in [5, 72] demonstriert. Eine Verwendung von Dienst-spezifischen Netzen für den Anwendungsfall Smart Metering wurde in [122] präsentiert. Eine Anwendungsschnittstelle für Dienst-spezifische Netze wurde in [71, 68] vorgestellt. Die Evaluation von Dienst-spezifischen Protokollen und Netzen wurde in [69, 70, 124] behandelt.

1.3 Gliederung

Diese Arbeit ist wie folgt gegliedert. In Kapitel 2 werden Dienst-spezifische Netze näher betrachtet und Begriffe eingeführt. Dabei wird gezeigt, dass für eine tatsächliche Nutzung Dienst-spezifischer Netze ein Zugriffsverfahren auf Dienst-spezifische Netze für Endsysteme, eine Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen auf einem Endsystem und ein leistungsfähiges Rahmenwerk für Dienst-spezifische Netze auf Endsystemen benötigt werden. In Kapitel 3 wird ein flexibles Zugriffsverfahren auf Dienst-spezifische Netze für Endsysteme vorgestellt. In Kapitel 4 wird ein flexibles Verwaltungssystem für die Netz-, Speicher- und Prozessor-Ressourcen auf einem Endsystem, die für Dienst-spezifische Netze verwendet werden, vorgestellt. In Kapitel 5 werden die Grundlagen und Analyse für Kapitel 6 vorgestellt. In Kapitel 6 wird ein leistungsfähiges Rahmenwerk für Dienst-spezifische Netze auf Endsystemen vorgestellt. Abschließend werden in Kapitel 7 die Ergebnisse dieser Arbeit zusammengefasst und ein Ausblick auf zukünftige Arbeiten vorgestellt.

Kapitel 2

Dienst-spezifische Netze

In diesem Kapitel werden die in dieser Arbeit verwendeten Dienst-spezifischen Netze näher vorgestellt und Begriffe eingeführt. In Abschnitt 2.1 wird eine Übersicht über Dienst-spezifische Netze und über die an deren Realisierung und Nutzung beteiligten Entitäten gegeben. In den darauf folgenden Abschnitten werden die Komponenten und Akteure Dienst-spezifischer Netze näher beschrieben. In Abschnitt 2.2 wird die Netz-Infrastruktur und die Konnektivität über diese vorgestellt. In Abschnitt 2.3 wird näher auf Dienst-spezifische Netze eingegangen. In Abschnitt 2.4 werden Endsysteme genauer betrachtet. In Abschnitt 2.5 werden der Entwurf und die Ausbringung Dienst-spezifischer Netze behandelt. In Abschnitt 2.6 wird die Interaktion zwischen Endsystemen und Dienst-spezifischen Netzen beschrieben. In Abschnitt 2.7 werden schließlich Herausforderungen gezeigt, für die im Rahmen dieser Arbeit Lösungen entwickelt wurden.

2.1 Übersicht

In diesem Abschnitt werden Dienst-spezifische Netze vorgestellt und grundlegende Begriffe eingeführt. Es wird gezeigt, welche Entitäten an der Realisierung und Nutzung Dienst-spezifischer Netze beteiligt sind und wie diese miteinander interagieren. Die Entitäten sind die folgenden:

- Technische Komponenten
- Akteure

Technische Komponenten sind die Komponenten, die für die Realisierung und Nutzung Dienst-spezifischer Netze verwendet werden. Akteure sind beispielsweise Personen oder Organisationen, die die technischen Komponenten verwalten oder nutzen und somit in der Ausbringung, dem Betrieb und der Nutzung Dienst-spezifischer Netze involviert

sind. Die wichtigsten technischen Komponenten sind die Dienst-spezifischen Netze selbst, die Infrastruktur-Domänen und die Endsysteme. *Dienst-spezifische Netze* sind logische Netze, die oberhalb der Netz-Infrastruktur ausgebracht werden. In ihnen werden Dienste mit Hilfe an den jeweiligen Dienst angepasster Netzstruktur, Protokolle und Anwendungen bereit gestellt. Die Netz-Infrastruktur besteht aus *Infrastruktur-Domänen*, in denen die Dienst-spezifischen Netze ausgebracht werden und die Konnektivität zu den Dienst-spezifischen Netzen ermöglichen. *Endsysteme* sind an Infrastruktur-Domänen angeschlossen und verbinden sich zu Dienst-spezifischen Netzen, um die darin angebotenen Dienste zu nutzen. Die Akteure im Netz-Modell sind Dienste-Anbieter, Anwendungsentwickler, Netz-Architekten, Virtuelle Netz-Betreiber, Infrastruktur-Betreiber und Nutzer. *Dienste-Anbieter* betreiben Dienst-spezifische Netze, um darin Dienste anzubieten. Für den Entwurf, die Ausbringung und den Betrieb Dienst-spezifischer Netze kooperieren Dienste-Anbieter mit weiteren Akteuren. *Anwendungsentwickler* entwerfen Anwendungen, die innerhalb Dienst-spezifischer Netze betrieben werden. *Netz-Architekten* sind für den Entwurf Dienst-spezifischer Netze und der darin verwendeten Protokolle verantwortlich. *Virtuelle Netz-Betreiber* bringen Dienst-spezifische Netze über der Netz-Infrastruktur aus. Hierfür kooperieren Virtuelle Netz-Betreiber wiederum mit Infrastruktur-Betreibern. *Infrastruktur-Betreiber* unterhalten eigene Infrastruktur-Domänen und sind für deren Betrieb verantwortlich. *Nutzer* verwenden Endsysteme bzw. Anwendungen auf ihnen, um auf Dienste in Dienst-spezifischen Netzen zuzugreifen.

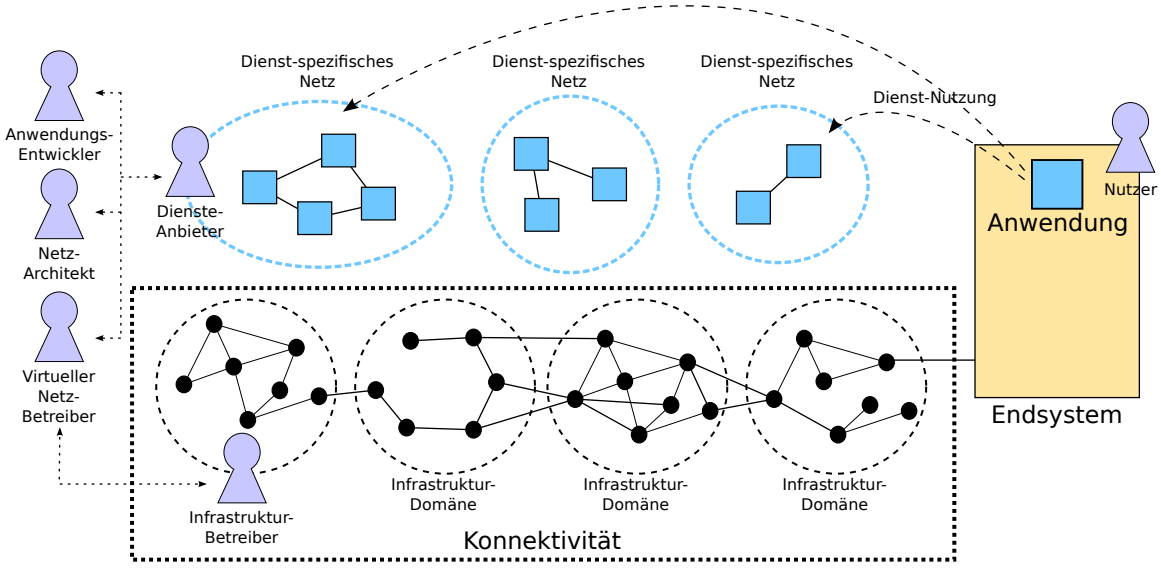

Abbildung 2.1: *Netz-Modell*

In Abbildung 2.1 wird das angenommene Netz-Modell mit den bereits erwähnten technischen Komponenten und Akteuren anhand eines Beispiels dargestellt. Die Netz-Infrastruktur besteht aus vier verschiedenen Infrastruktur-Domänen. Die Infrastruktur-

Domänen bestehen wiederum aus Infrastruktur-Knoten, die über Infrastruktur-Verbindungen miteinander verbunden sind. Stellvertretend für alle Domänen in diesem Beispiel ist an einer Infrastruktur-Domäne ein Infrastruktur-Betreiber eingezeichnet. Dieser ist für den Betrieb der jeweiligen Infrastruktur-Domäne verantwortlich. Oberhalb der Netz-Infrastruktur sind drei Dienst-spezifische Netze ausgebracht. Diese stellen jeweils Dienste bereit. Stellvertretend für alle Dienst-spezifischen Netze in diesem Beispiel ist bei einem ein Dienste-Anbieter eingezeichnet. Dieser ist für den Betrieb des entsprechenden Dienst-spezifischen Netzes verantwortlich. Der Dienste-Anbieter kann, wie im Beispiel dargestellt, für die Ausbringung und den Betrieb des Dienst-spezifischen Netzes mit einem Anwendungsentwickler, einem Netz-Architekten und einen Virtuellen Netz-Betreiber interagieren. Der Virtuelle Netz-Betreiber interagiert wiederum mit Infrastruktur-Betreibern, um das Dienst-spezifische Netz über die Netz-Infrastruktur aufbauen zu können. An eine Infrastruktur-Domäne kann ein Endsystem angeschlossen sein (vgl. Infrastruktur-Domäne rechts in Abbildung 2.1). Auf dem Endsystem verwendet in diesem Beispiel ein Nutzer eine Anwendung, um auf die Dienste, die in zwei der drei vorhandenen Dienst-spezifischen Netze angeboten werden, zuzugreifen.

Im Folgenden werden die vorhandenen Komponenten und beteiligten Akteure der Netz-Infrastruktur, der Dienst-spezifischen Netze und der Endsysteme genauer beschrieben. Anschließend wird näher auf den Aufbau Dienst-spezifischer Netze und auf die Interaktion zwischen Endsystemen und Dienst-spezifischen Netzen eingegangen.

2.2 Netz-Infrastruktur und Konnektivität

Wie in Abbildung 2.1 gezeigt, besteht die Netz-Infrastruktur aus einem Zusammenschluss verschiedener Infrastruktur-Domänen. Eine Infrastruktur-Domäne ist ein Teil der Netz-Infrastruktur, der unter der administrativen Kontrolle eines Infrastruktur-Betreibers steht. Ein Infrastruktur-Betreiber ist somit eine natürliche Person oder eine Organisation, die eine eigene Infrastruktur betreibt. Die Infrastruktur-Domäne besteht aus Infrastruktur-Knoten und Infrastruktur-Verbindungen zwischen diesen Knoten. Infrastruktur-Knoten sind z.B. Netzgeräte wie Router und Switches, Server- und Speichersysteme. Die Infrastruktur-Verbindungen sind zum Beispiel die Verkabelung zwischen den Netzgeräten. Der Infrastruktur-Betreiber hat vollen Zugriff auf und Kontrolle über die Infrastruktur-Knoten und Infrastruktur-Verbindungen in seiner Domäne. Somit kann er die Infrastruktur-Domäne an seine Bedürfnisse anpassen. Zum Beispiel kann der Betreiber die Struktur des Netzes und die auf den Knoten verfügbaren Protokolle frei wählen. Die Protokolle, die in einer Infrastruktur-Domäne verwendet werden, werden durch einen Infrastruktur-Protokoll-Stapel bereit gestellt, der auf den Infrastruktur-Knoten installiert ist. Ein Beispiel für einen Infrastruktur-Protokoll-Stapel ist ein Protokoll-Stapel bestehend aus den Protokollen TCP [89] und UDP [90], die über das Protokoll IPv4 [88] betrieben werden,

welches wiederum über Ethernet [46] kommuniziert. Dies entspricht einem gängigen Protokoll-Stapel im heutigen Internet. Verschiedene Infrastruktur-Domänen werden mit Hilfe von Infrastruktur-Verbindungen miteinander verbunden. Durch die Unterstützung gemeinsamer Protokolle innerhalb der Infrastruktur-Protokoll-Stapel der miteinander verbundenen Infrastruktur-Domänen können die Infrastruktur-Betreiber eine Domänenübergreifende Konnektivität ermöglichen. Im heutigen Internet wird zum Beispiel das Protokoll IPv4 als ein solches gemeinsames Protokoll verwendet, um Konnektivität über das Internet zu erreichen.

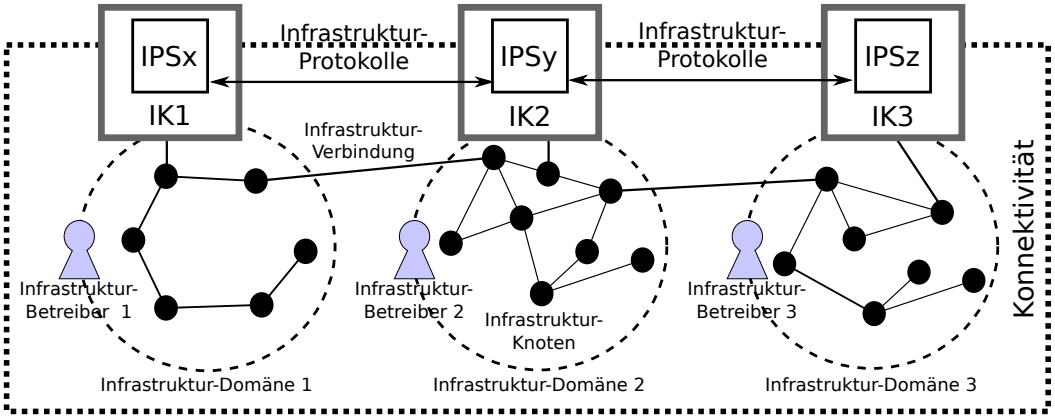

Abbildung 2.2: *Netz-Infrastruktur*

In Abbildung 2.2 wird die Netz-Infrastruktur detaillierter anhand eines Beispiels dargestellt. Drei Infrastruktur-Domänen sind miteinander über Infrastruktur-Verbindungen verbunden. Jede Infrastruktur-Domäne wird von einem Infrastruktur-Betreiber betrieben. Beispielhaft ist in jeder Infrastruktur-Domäne ein Infrastruktur-Knoten (*IK*) hervorgehoben und genauer dargestellt. Jeder Infrastruktur-Knoten betreibt einen Infrastruktur-Protokoll-Stapel (*IPS*), den der Infrastruktur-Betreiber in seiner Infrastruktur-Domäne vorgesehen hat. Die Infrastruktur-Knoten in der selben Infrastruktur-Domäne können über die Infrastruktur-Protokolle in ihrem IPS miteinander kommunizieren. Infrastruktur-Knoten unterschiedlicher Infrastruktur-Domänen können miteinander kommunizieren, sofern die Domänen über Infrastruktur-Verbindungen miteinander verbunden sind und ihre IPS gemeinsame Protokolle enthalten, die den Austausch von Nachrichten ermöglichen. Zum Beispiel kann der Knoten *IK1* mit Knoten *IK2* kommunizieren, sofern *IPSx* und *IPSy* ein gemeinsames Protokoll enthalten. Indem alle IPS in diesem Beispiel gemeinsame Protokolle enthalten kann eine Konnektivität zwischen beliebigen Infrastruktur-Knoten beliebiger Infrastruktur-Domänen erreicht werden.

2.3 Dienst-spezifische Netze

Ein Dienst-spezifisches Netz ist ein logisches Netz oberhalb der Netz-Infrastruktur, das einen Dienst bereit stellt und an die Anforderungen dieses Dienstes angepasst ist. Ein Dienst ist dabei eine Leistung, die Endsystemen bzw. Nutzern zur Verfügung gestellt wird. Beispiele für Dienste sind Video- und Musik-Streaming, Zwischenspeichern von Inhalten auf dem Datenpfad (Caching) oder Informationsdienste wie (verteilte) Datenbanken. Dienste, wie zum Beispiel ein Download-Dienst, können Inhalte bereitstellen. Inhalte sind zum Beispiel Dateien, Text-Dokumente oder Musik-Stücke. Ein Dienst-spezifisches Netz steht unter der Kontrolle eines Dienste-Anbieters. Ein Dienste-Anbieter ist eine natürliche Person oder Organisation, die Nutzern Dienste anbietet und hierfür ein Dienst-spezifisches Netz betreibt. Netz-Virtualisierung wird verwendet, um das Dienst-spezifische Netz oberhalb der Netz-Infrastruktur aufzubauen. Das Dienst-spezifische Netz besteht daher aus Virtuellen Knoten, die über Virtuelle Verbindungen miteinander verbunden werden. Außerdem wird auf den Knoten eines Dienst-spezifischen Netzes ein Dienst-spezifischer Protokoll-Stapel ausgeführt. Dieser Protokoll-Stapel enthält Protokolle, die an die Erbringung des Dienstes und an das Dienst-spezifische Netz angepasst sind. Zusätzlich werden auf den Knoten Dienst-spezifische Anwendungen ausgeführt, die zur Erbringung des Dienstes genutzt werden. Hierbei kann es sich einerseits um verschiedene Instanzen einer verteilten Anwendung handeln. Andererseits sind verschiedene Anwendungen möglich, die miteinander oder mit den Endsystemen zur Erbringung des Dienstes kommunizieren.

Der Dienste-Anbieter interagiert mit weiteren Akteuren, um Dienste bzw. Dienste-spezifische Netze zu realisieren. Anwendungsentwickler sind natürliche Personen oder Organisationen, die für Dienste-Anbieter die zur Erbringung eines Dienstes benötigten Dienst-spezifischen Anwendungen entwickeln. Hierbei handelt es sich um die Anwendungen, die innerhalb des Dienst-spezifischen Netzes ausgebracht werden. Darüber hinaus können Anwendungsentwickler auch Anwendungen entwickeln, die auf Endsystemen für die Nutzung von Diensten verwendet werden. Dies kann zum Beispiel der Fall sein, wenn die Anwendung auf den Endsystemen zur Dienst-Nutzung ein Teil einer verteilten Anwendung im Dienst-spezifischen Netz sein soll. Netz-Architekten sind natürliche Personen oder Organisationen, die für Dienste-Anbieter die Dienst-spezifischen Netze entwerfen. Dabei sind sie für die Struktur und Eigenschaften der Netze sowie die darin verwendeten Protokolle verantwortlich. Virtuelle Netz-Betreiber sind natürliche Personen oder Organisationen, die den Dienste-Anbietern die für Dienst-spezifische Netze benötigten virtuellen Netze zur Verfügung stellen. Damit die Virtuellen Netz-Betreiber virtuelle Netze über die Netz-Infrastruktur aufbauen können, interagieren sie mit Infrastruktur-Betreibern.

Abbildung 2.3 stellt den Aufbau eines Dienst-spezifischen Netzes anhand eines Beispiels genauer dar. Das Dienst-spezifische Netz wird von einem Dienste-Anbieter betrieben. Es besteht aus drei Virtuellen Knoten, die jeweils über Virtuelle Verbindungen miteinander

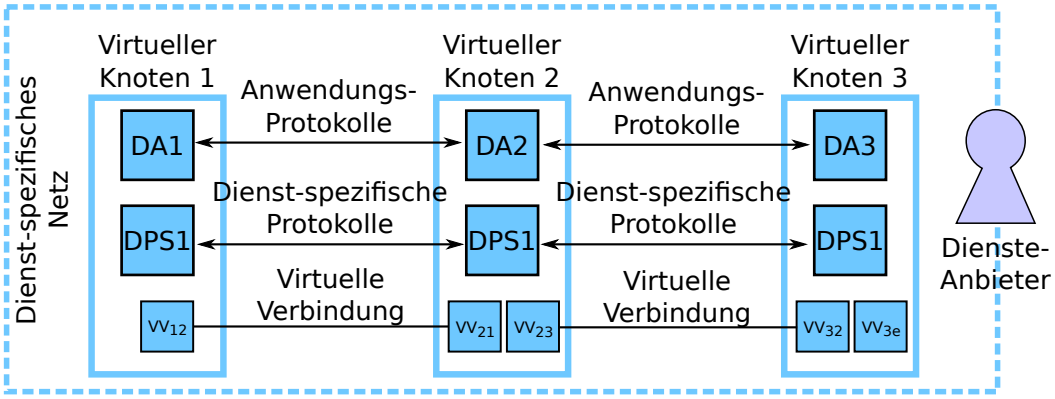

Abbildung 2.3: *Dienst-spezifisches Netz*

verbunden sind. Eine Virtuelle Verbindung von Knoten *x* zu Knoten *y* wird auf Knoten *x* durch ihren Endpunkt *VVxy* repräsentiert. Auf jedem Virtuellen Knoten wird der Dienst-spezifische Protokoll-Stapel *DPS1* betrieben. Auf den Virtuellen Knoten werden zudem die Dienst-spezifischen Anwendungen *DA1*, *DA2* und *DA3* ausgeführt. Dabei wird *DA1* auf Knoten 1, *DA2* auf Knoten 2 und *DA3* auf Knoten 3 betrieben. Die Anwendungen kommunizieren miteinander über Anwendungsprotokolle. Die Nachrichten der Anwendungsprotokolle werden über die Dienst-spezifischen Protokolle des Dienst-spezifischen Protokollstapels transportiert.

2.3.1 Beispiel

Die Realisierung und Nutzung eines Dienstes in einem Dienst-spezifischen Netz wird anhand des Anwendungsfalles in Abbildung 2.4 gezeigt. Hierbei handelt es sich um ein Dienst-spezifisches Netz für einen Concast-Dienst wie z.B. Smart-Metering [122].

Dieses Dienst-spezifische Netz besitzt eine baumförmige Netzstruktur. Die Virtuellen Knoten *VK1* bis *VK6* und die Virtuelle Verbindungen, die sie miteinander verbinden, bilden also einen Baum. Die Endsysteme können sich wie das Endsystem im Beispiel nur mit den Virtuellen Knoten verbinden, die die Blätter der Baumstruktur repräsentieren. Die Endsysteme dienen als Datenquellen. Der Virtuelle Knoten, der sich an der Wurzel des Baumes befindet, dient als Datensenke. Die von der Anwendung *Ax* in den Endsystemen generierten Daten werden über die Protokolle des Dienst-spezifischen Protokoll-Stapels *DPS1* von den Endsystemen zu den Virtuellen Knoten und anschließend von Virtuellem Knoten zu Virtuellem Knoten entlang der Baumstruktur in Richtung der Wurzel weitergeleitet. Dabei werden die Daten aggregiert. Treffen die Daten an der Datensenke ein, werden sie auf diesem Virtuellen Knoten von der Dienst-spezifischen Anwendung *DA1* verarbeitet. Auf den anderen Virtuellen Knoten in diesem Beispiel werden keine Dienst-spezifischen Anwendungen benötigt.

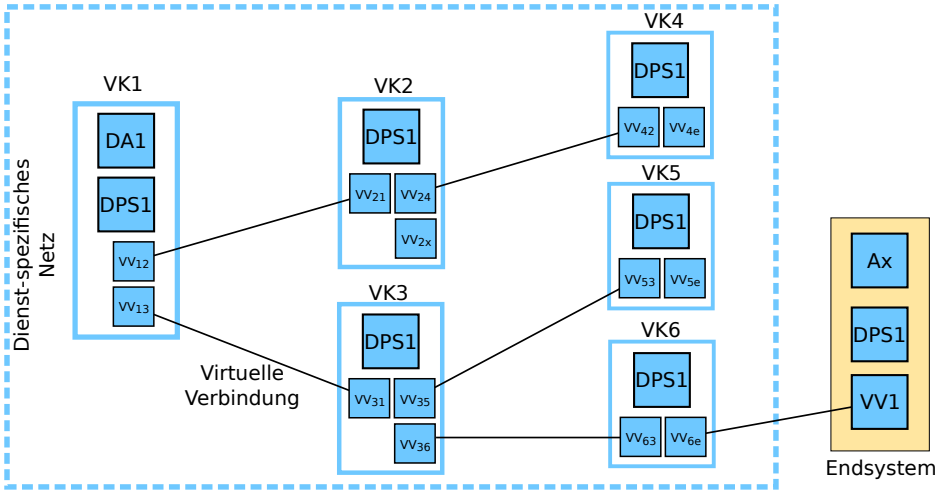

Abbildung 2.4: *Dienst-spezifisches Netz für einen Concast-Dienst wie z.B. Smart-Metering.*

Dank der Netzstruktur kann ein statisches Routing verwendet werden, das Dateneinheiten entlang des Baumes in Richtung der Wurzel weiterleitet. Für die Übertragung der Daten kann ein Protokoll verwendet werden, das über einen festgelegten Zeitraum eintreffende Dateneinheiten puffert, aggregiert und die resultierende Dateneinheit nach Ablauf des Zeitraumes weiterleitet.

2.4 Endsysteme

Ein Endsystem ist ein Gerät, das einem Nutzer zur Verfügung steht und das dieser verwendet, um auf Dienste zuzugreifen. Ein Nutzer ist eine natürlichen Person, die die in Dienste-spezifischen Netzen bereitgestellten Dienste nutzt. Ein Endsystem ist z.B. ein Computer, Tablet oder Smartphone bestehend aus gängiger Hardware, auf dem ein Betriebssystem wie z.B. Linux oder Windows verwendet wird. Ein Endsystem ist über (mindestens) eine Infrastruktur-Verbindung mit (mindestens) einer Infrastruktur-Domäne verbunden. Auf dem Endsystem werden daher auch die entsprechenden Infrastruktur-Protokoll-Stapel ausgeführt. Für die Nutzung von Diensten unterhält das Endsystem zudem Virtuelle Verbindungen zu Dienst-spezifischen Netzen. Für diese Dienst-spezifischen Netze führt das Endsystem Dienst-spezifische Protokoll-Stapel aus. Außerdem werden auf dem Endsystem die Anwendungen ausgeführt, die der Nutzer für den Zugriff auf Dienste verwendet. Eine Anwendung kann dabei auf Dienste in verschiedenen Dienst-spezifischen Netzen zugreifen. Beispiele für eine solche Anwendung sind Webbrowser oder Download-Manager zum Herunterladen von Dateien. Eine Anwendung kann allerdings auch an die Nutzung spezieller Dienste in speziellen Dienst-spezifischen Netzen

angepasst werden. Beispiele für solche Anwendungen sind spezielle Peer-to-Peer oder Video-Streaming Anwendungen.

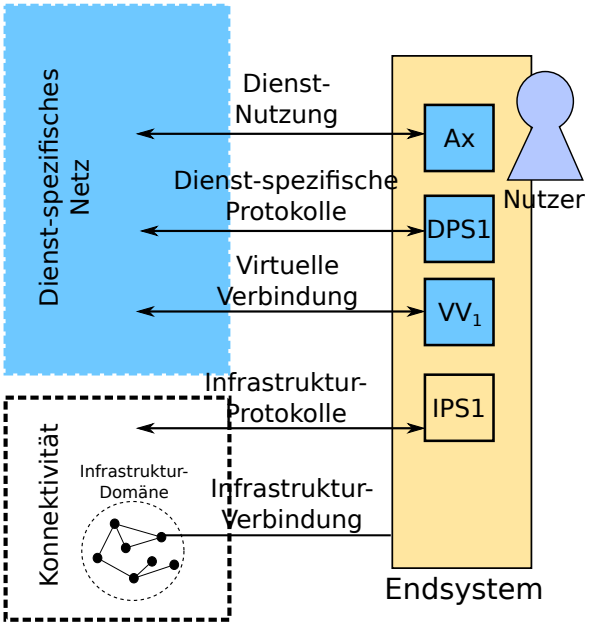

Abbildung 2.5: *Endsystem*

Abbildung 2.5 stellt den Aufbau eines Endsystems anhand eines Beispiels detaillierter dar. Das Endsystem ist über eine Infrastruktur-Verbindung mit einer Infrastruktur-Domäne verbunden. Daher wird auf dem Endsystem der Infrastruktur-Protokoll-Stapel dieser Infrastruktur-Domäne (*IPS1*) betrieben. Der Infrastruktur-Protokoll-Stapel enthält die Infrastruktur-Protokolle, mit denen das Endsystem mit anderen Infrastruktur-Knoten kommunizieren kann. Außerdem betreibt das Endsystem eine Virtuelle Verbindung (*VV1*) zu einem Dienst-spezifischen Netz. Für dieses Dienst-spezifische Netz führt das Endsystem zudem den Dienst-spezifischen Protokoll-Stapel *DPS1* aus, der die Dienst-spezifischen Protokolle des Dienst-spezifischen Netzes enthält. Außerdem führt das Endsystem eine Anwendung (*Ax*) aus, die der Nutzer des Endsystems benutzt, um auf den Dienst, der über das Dienst-spezifische Netz angeboten wird, zuzugreifen.

2.5 Entwurf und Ausbringung Dienst-spezifischer Netze

In diesem Abschnitt wird näher auf den Aufbau Dienst-spezifischer Netze eingegangen. Es wird gezeigt, wie Dienst-spezifische Netze entworfen und über der Netz-Infrastruktur ausgebracht werden. Dabei wird auch diskutiert, wie die Kommunikation zwischen den Knoten des Dienst-spezifischen Netzes realisiert wird.

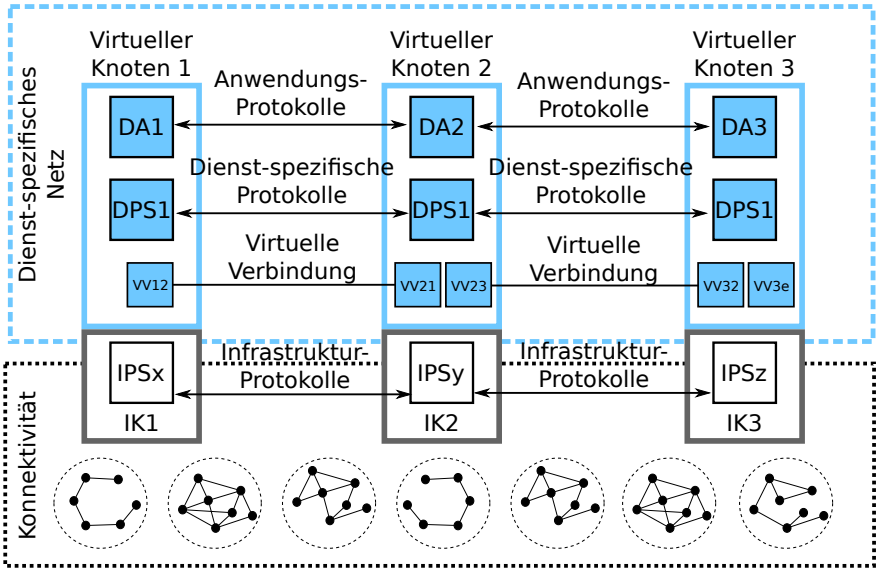

Abbildung 2.6: *Aufbau eines Dienst-spezifischen Netzes.*

In Abbildung 2.6 wird der Aufbau Dienst-spezifischer Netze anhand eines Beispiels dargestellt. Die Netz-Infrastruktur besteht aus sieben Infrastruktur-Domänen. Die Infrastruktur-Domänen bestehen wiederum aus Infrastruktur-Knoten die über Infrastruktur-Verbindungen miteinander verbunden sind. Wie die Infrastruktur-Domänen miteinander verbunden sind, ist in diesem Beispiel nicht näher spezifiziert. Es wird allerdings angenommen, dass über die Infrastruktur-Domänen Konnektivität zwischen Infrastruktur-Knoten ermöglicht wird. Oberhalb der Netz-Infrastruktur ist ein Dienst-spezifisches Netz ausgebracht. Es besteht aus drei Virtuellen Knoten, die über Virtuelle Verbindungen (*VV*) miteinander verbunden sind. Auf jedem Knoten wird der Dienst-spezifische Protokoll-Stapel *DPS1* ausgeführt. Außerdem befinden sich auf den Virtuellen Knoten die Dienst-spezifischen Anwendungen *DA1* bis *DA3*, wobei auf jedem Knoten jeweils nur eine Anwendung ausgeführt wird. Die Virtuellen Knoten sind auf den drei Infrastruktur-Knoten *IK1* bis *IK3* ausgebracht. Diese sind Teil nicht näher spezifizierter Infrastruktur-Domänen und verwenden die Infrastruktur-Protokoll-Stapel *IPSx*, *IPSy* und *IPSz*. Innerhalb des Dienst-spezifischen Netzes können die Anwendungen über Anwendungsprotokolle miteinander kommunizieren. Diese Anwendungsprotokolle nutzen die Dienst-spezifischen Protokolle des Dienst-spezifischen Protokoll-Stapels zur Kommunikation. Die Virtuellen Verbindungen zwischen den Virtuellen Knoten werden über die Infrastruktur-Protokolle, die in den Infrastruktur-Protokoll-Stapeln auf den Infrastruktur-Knoten unterstützt werden, etabliert. Die zwischen den jeweiligen Anwendungen, Virtuellen Knoten und Infrastruktur-Knoten verwendeten Protokolle können unterschiedlich sein. Die für die Virtuellen Verbindungen verwendeten Protokolle hängen davon ab, welche Protokolle für die Kommunikation zwischen den Infrastruktur-Knoten *IK1* bis

IK3 verfügbar sind. Die für die Kommunikation oberhalb der Virtuellen Verbindungen verwendeten Protokolle sind dienst- bzw. anwendungsabhängig.

2.5.1 Entwurf

Der Entwurf eines Dienst-spezifischen Netzes ist in eine Reihe von zusammenhängenden Aufgaben untergliedert: Der Entwurf des Dienstes, der Entwurf der Netzstruktur und der Entwurf der Protokolle. Diese Aufgaben sind zusammenhängend, da sie sich, wie im Folgenden näher beschrieben, gegenseitig beeinflussen. Bei diesen Aufgaben interagiert der Dienste-Anbieter außerdem mit verschiedenen anderen Akteuren.

Beim Entwurf des Dienstes interagiert der Dienste-Anbieter mit dem Anwendungsentwickler. Der Anwendungsentwickler entwirft die Anwendungen und Anwendungsprotokolle, die für die Erbringung des Dienstes über das Dienst-spezifische Netz benötigt werden. Der Entwurf der Anwendungen und der Anwendungsprotokolle basiert dabei auf den Eigenschaften und Anforderungen des Dienstes, der geplanten Netzstruktur des Dienst-spezifischen Netzes und den geplanten Dienst-spezifischen Protokollen.

Beim Entwurf der Netzstruktur interagiert der Dienste-Anbieter mit dem Netz-Architekten. Dieser legt die Struktur des Dienst-spezifischen Netzes basierend auf den Eigenschaften und Anforderungen des Dienstes, den geplanten Anwendungen und den geplanten Dienst-spezifischen Protokollen fest. Hierbei bestimmt der Netz-Architekt die Anzahl Virtueller Knoten und die Virtuellen Verbindungen zwischen ihnen. Darüber hinaus legt er die Eigenschaften der Virtuellen Knoten und Eigenschaften der Virtuellen Verbindungen fest.

Beim Entwurf der Dienst-spezifischen Protokolle interagiert der Dienste-Anbieter wiederum mit dem Netz-Architekten. Dieser wählt existierende oder entwirft neue Protokolle, die innerhalb des Dienst-spezifischen Netzes benutzt werden, basierend auf den Eigenschaften und Anforderungen des Dienstes, den geplanten Anwendungen und der geplanten Netzstruktur.

2.5.2 Ausbringung

Nach dem Entwurf des Dienst-spezifischen Netzes wird es ausgebracht. Hierzu interagiert der Dienste-Anbieter mit dem Virtuellen Netz-Betreiber. Der Virtuelle Netz-Betreiber bringt das entworfene Dienst-spezifische Netz über der Netz-Infrastruktur aus. Dabei interagiert der Virtuelle Netz-Betreiber mit den Infrastruktur-Betreibern, deren Infrastruktur er verwendet. Diese zusätzliche Interaktion wird allerdings nicht näher betrachtet.

Der Virtuelle Netz-Betreiber instantiiert die Virtuellen Knoten auf zuvor ausgewählten Infrastruktur-Knoten. Bei der Auswahl der Infrastruktur-Knoten muss der Virtuelle Netz-Betreiber dafür sorgen, dass die angeforderten Eigenschaften des Virtuellen Knoten auf den Infrastruktur-Knoten gewährleistet werden. Anschließend instantiiert der Virtuelle

Netz-Betreiber die Virtuellen Verbindungen zwischen den Virtuellen Knoten über die Netz-Infrastruktur. Auch hierbei muss der Virtuelle Netz-Betreiber dafür sorgen, dass die angeforderten Eigenschaften der Virtuellen Verbindungen auf den Infrastruktur-Knoten und auf den Datenpfaden zwischen diesen eingehalten werden.

Sobald das Dienst-spezifische Netz instantiiert ist, kann der Dienste-Anbieter mit der Ausbringen der Komponenten des Dienst-spezifischen Netzes auf den Virtuellen Knoten beginnen. Hierbei bringt der Dienste-Anbieter, wie im Entwurf festgelegt, die Dienst-spezifischen Protokolle und Anwendungen auf den Virtuellen Knoten aus. Abschließend konfiguriert der Dienste-Anbieter die Anwendungen und Dienst-spezifischen Protokolle auf den Virtuellen Knoten. Nach der Ausbringung des Dienst-spezifischen Netzes kann der Dienste-Anbieter es verwenden, um Nutzern den entsprechenden Dienst zur Verfügung zu stellen.

2.6 Interaktion zwischen Endsystem und Dienst-spezifischem Netz

Damit ein Nutzer einen Dienst, der in einem Dienst-spezifischen Netz angeboten wird, nutzen kann, interagieren Komponenten auf dem Endsystem des Nutzers, Komponenten in der Netz-Infrastruktur und Komponenten des Dienst-spezifischen Netzes miteinander. Bei der Interaktion der Komponenten werden verschiedene Protokolle verwendet. In Abbildung 2.7 werden die Interaktion und die Protokolle anhand eines Beispiels dargestellt.

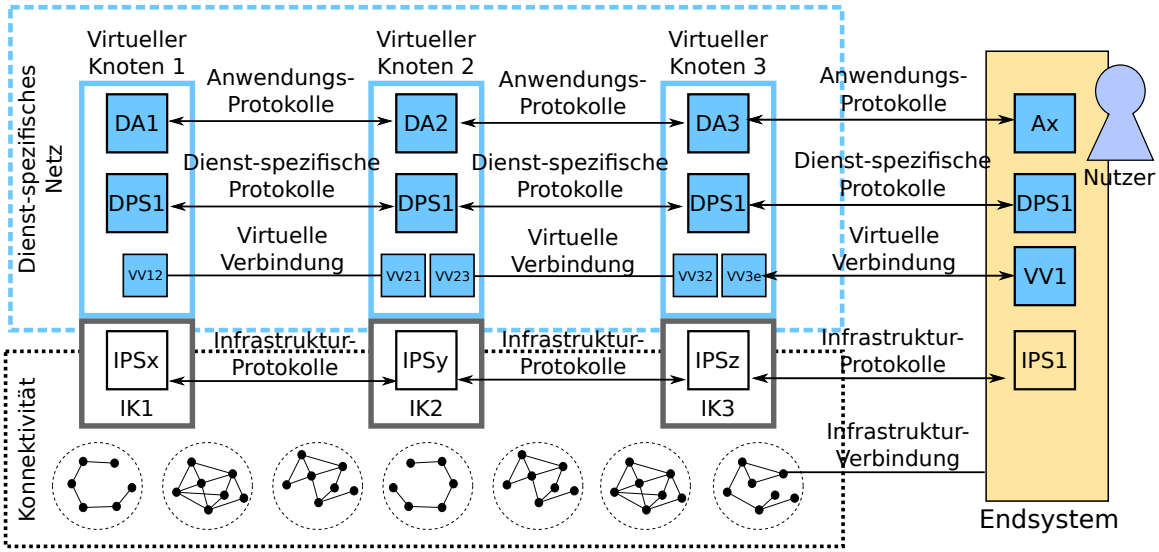

Abbildung 2.7: *Interaktion zwischen Nutzer und Dienst-spezifischem Netz.*

In diesem Beispiel wird eine identische Netz-Infrastruktur und ein identisches Dienst-spezifisches Netz wie im vorherigen Beispiel (siehe Abbildung 2.6) verwendet. Die Netz-Infrastruktur besteht wieder aus sieben Infrastruktur-Domänen. Wie diese miteinander verbunden sind, ist auch in diesem Beispiel nicht näher spezifiziert. Es wird allerdings angenommen, dass über die Infrastruktur-Domänen Konnektivität zwischen Infrastruktur-Knoten und Endsystemen ermöglicht wird. An eine Infrastruktur-Domäne ist ein Endsystem angeschlossen. Auf dem Endsystem ist ein Infrastruktur-Protokoll-Stapel *IPS1* installiert. Darüber hinaus betreibt das Endsystem eine Virtuelle Verbindung (*VV1*) zum Dienst-spezifischen Netz und den Dienst-spezifischen Protokoll-Stapel *DPS1*. Auf dem Endsystem nutzt die Anwendung *Ax* den durch das Dienst-spezifische Netz bereitgestellten Dienst und interagiert mit dem Nutzer. Die Anwendung *Ax* kommuniziert mit den Dienst-spezifischen Anwendungen im Dienst-spezifischen Netz über Anwendungsprotokolle. Diese Protokolle regeln Abläufe innerhalb der Anwendungen. Die Anwendungsprotokolle kommunizieren über die Dienst-spezifischen Protokolle, die durch den Dienst-spezifischen Protokoll-Stapel *DPS1* bereit gestellt werden. Diese Protokolle ermöglichen die Kommunikation innerhalb des Dienst-spezifischen Netzes. Die Dienst-spezifischen Protokolle kommunizieren wiederum über die Virtuelle Verbindung. Die Virtuelle Verbindung wird über die Netz-Infrastruktur aufgebaut. Daher wird sie über die Infrastruktur-Protokolle betrieben, die durch den Infrastruktur-Protokoll-Stapel bereit gestellt werden und zur Kommunikation zwischen dem Endsystem und dem Infrastruktur-Knoten zur Verfügung stehen.

Die zwischen den jeweiligen Anwendungen, Virtuellen Knoten, Infrastruktur-Knoten und dem Endsystem verwendeten Protokolle können unterschiedlich sein. Die für die Virtuellen Verbindungen verwendeten Protokolle hängen davon ab, welche Protokolle für die Kommunikation zwischen den Infrastruktur-Knoten *IK1* bis *IK3* bzw. zwischen dem Endsystem und dem Infrastruktur-Knoten *IK3* verfügbar sind. Die für die Kommunikation oberhalb der Virtuellen Verbindungen verwendeten Protokolle sind Dienst- bzw. anwendungsabhängig. Ebenso ist das Muster, nach welchem die Kommunikation zwischen dem Endsystem und den Virtuellen Knoten innerhalb des Dienst-spezifischen Netz abläuft, abhängig vom Dienst, von den Anwendungen und von der Struktur sowie den Protokollen des Dienst-spezifischen Netzes. Beispielsweise können in einem Dienst-spezifischen Netz Routing-Protokolle und Weiterleitungsmechanismen im Dienst-spezifischen Protokoll-Stapel eine Kommunikation zwischen den Endsystemen und beliebigen Virtuellen Knoten erlauben. Ein anderes Dienst-spezifisches Netz kann so beschaffen sein, dass Endsysteme nur mit einem speziellen Virtuellen Knoten mit Hilfe Dienst-spezifischer Protokolle kommunizieren können. Dieser Knoten bzw. die Anwendung darauf kommuniziert stellvertretend für Endsysteme mit anderen Knoten bzw. Anwendungen im Dienst-spezifischen Netz über möglicherweise andere Dienst-spezifische Protokolle, um den Dienst dem Endsystem bereit zu stellen.

2.7 Herausforderungen

Die in diesem Kapitel vorgestellten Konzepte Dienst-spezifischer Netze werfen eine Reihe von Fragestellungen auf. Drei der Herausforderungen Dienst-spezifischer Netze werden in Abbildung 2.8 aufgezeigt, für die im Rahmen dieser Arbeit Lösungen entworfen wurden. Diese drei Herausforderungen Dienst-spezifischer Netze sind: (1) Anbindung an Dienst-spezifische Netze, (2) Ressourcen-Verwaltung und (3) Schneller Datenaustausch.

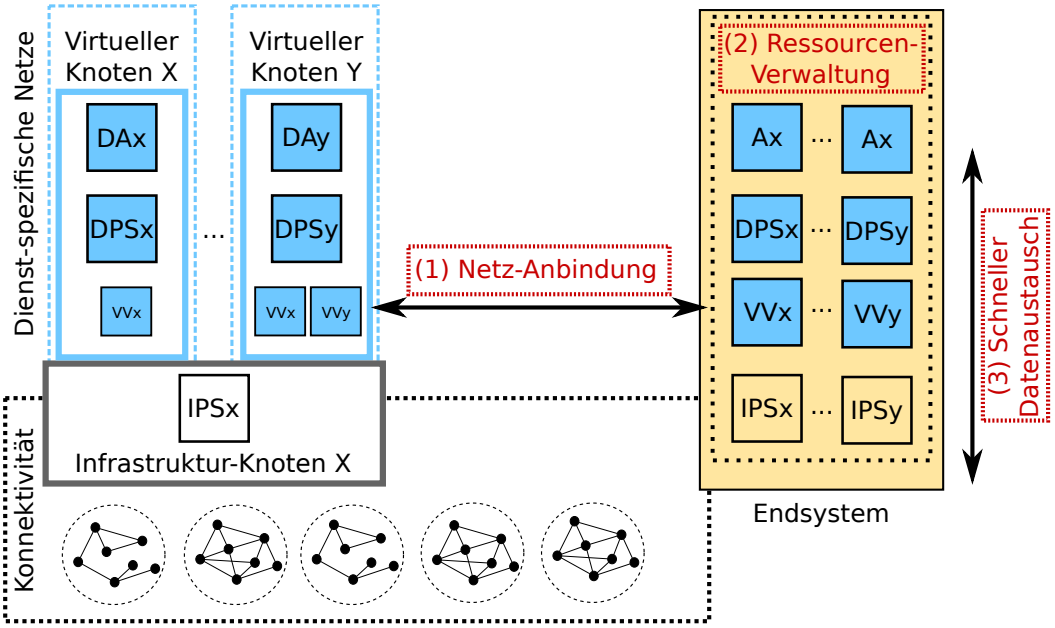

Abbildung 2.8: *Herausforderungen Dienst-spezifischer Netze: (1) Anbindung an Dienst-spezifische Netze, (2) Ressourcen-Verwaltung, (3) Schneller Datenaustausch*

Anbindung an Dienst-spezifische Netze

Wenn eine Anwendung auf einem Endsystem auf einen Dienst zugreift, der in einem oder mehreren Dienst-spezifischen Netzen angeboten wird, muss das Endsystem an ein entsprechendes Dienst-spezifisches Netz angebunden werden. Dabei müssen folgende Teilprobleme gelöst werden. Das Endsystem muss ermitteln, welche Dienst-spezifischen Netze den Dienst anbieten. Stehen mehrere Dienst-spezifische Netze zur Auswahl, muss das Endsystem eines der Dienst-spezifischen Netze auswählen. Das Endsystem muss außerdem ermitteln, wie eine Virtuelle Verbindung zu dem Dienst-spezifischen Netz aufgebaut werden kann, und schließlich eine Verbindung aufbauen. Damit das Endsystem über die Virtuelle Verbindung kommunizieren kann, muss das Endsystem zudem herausfinden, wie es den Dienst-spezifischen Protokoll-Stapel beziehen kann, und anschließend den Protokoll-Stapel beziehen.

Ressourcen-Verwaltung

Anwendungen, Dienst-spezifische Protokoll-Stapel und Virtuelle Verbindungen benötigen Prozessor-, Speicher- und Netzwerk-Ressourcen eines Endsystems. Dabei kann die Ressourcen-Nutzung der verschiedenen Komponenten auf einem Endsystem unterschiedlich ausfallen. Beispielsweise kann ein Protokoll-Stapel zur Verarbeitung von Paketen mehr CPU-Zeit verbrauchen als ein anderer, oder eine Anwendung kann eine höhere Datenrate verursachen als eine andere. Die Ressourcen des Endsystems sind allerdings beschränkt und die Komponenten können sich durch ihre Ressourcen-Nutzung negativ beeinflussen, wenn z.B. einer Komponente aufgrund anderer Komponenten nicht mehr genug Ressourcen zur Verfügung stehen. Eine Herausforderung Dienst-spezifischer Netze ist demnach, wie die Ressourcen eines Endsystems den auf dem Endsystem aktiven Komponenten zugewiesen werden können. Es wird also eine Ressourcen-Verwaltung benötigt, die die vorhandenen Ressourcen des Endsystems auf die verschiedenen Komponenten verteilt. Somit kann beispielsweise erreicht werden, dass die vorhandenen Ressourcen gleichmäßig auf die Komponenten verteilt werden. Im Falle, dass gewisse Komponenten als wichtiger als andere eingestuft werden, könnten außerdem diesen Komponenten mehr Ressourcen als anderen zugeteilt werden.

Schneller Datenaustausch

Soll ein Dienst über ein Dienst-spezifisches Netz durch ein Endsystem genutzt werden, ist eine hohe Leistung wünschenswert. Bei der Kommunikation sollten möglichst hohe Daten- und Paket-Raten erreicht werden. Eine Herausforderung Dienst-spezifischer Netze ist also das Erreichen eines schnellen Datenaustauschs zwischen Anwendungen, Dienst-spezifischen Protokoll-Stapeln und Virtuellen Verbindungen über die vorhandene Netz-Infrastruktur. Dabei sollte möglichst die Leistung vorhandener Infrastruktur-Verbindungen des Endsystems ausgenutzt werden können.

Kapitel 3

Zugriff von Endsystemen auf Dienst-spezifische Netze

Eine wichtige Fragestellung ist, wie ein Endsystem ohne Vorwissen auf ein Dienst-spezifisches Netz (DsN) zugreifen kann, um den darin angebotenen Dienst zu nutzen. Außerdem sollte der Zugriff sowohl über das heutige Internet als auch über zukünftige Kommunikationsnetze funktionieren. Damit ein Nutzer auf einem Endsystem einen Dienst, der in einem DsN angeboten wird, nutzen kann, muss das Endsystem auf das entsprechende DsN zugreifen. Ein Endsystem hat allerdings kein Vorwissen darüber, welche Dienste ein Nutzer während der Laufzeit verwenden wird. Daher müssen folgende Teilprobleme zur Laufzeit, wenn der Nutzer auf einen Dienst zugreifen will, gelöst werden: (1) Finden eines DsN für einen Dienst, (2) Beziehen der Protokolle, die im DsN verwendet werden, und (3) Verbinden zu dem DsN. In diesem Kapitel wird EUNA (*End User Node Access*), ein flexibles Zugriffsverfahren vorgestellt, das diese Probleme transparent für den Nutzer löst. EUNA verwendet einen Mapping-Dienst zum Ablegen von Informationen, einen Speicher-Dienst zum Speichern von Protokollen und ein Basisnetz für die Konnektivität zwischen Endsystemen, dem Mapping-Dienst und dem Speicher-Dienst. Greift der Nutzer auf einen Dienst zu, nutzt das Endsystem den Mapping-Dienst, um den vom Nutzer angeforderten Dienst in ein DsN, eine Liste von Zugangspunkten zum DsN und eine Liste von Protokollen aufzulösen. Anschließend bezieht das Endsystem die Protokolle aus dem Speicher-Dienst und baut eine Verbindung zum DsN auf. Diese Schritte werden von EUNA automatisiert und ohne Interaktion mit dem Nutzer durchgeführt. Der Nutzer benötigt daher kein Wissen über Netz-Details. Durch den modularen Aufbau der Lösung wird außerdem ein hoher Grad an Flexibilität erreicht. Es ist möglich, sowohl die Methoden, die für das Beziehen der Dienst-spezifischen Protokolle eingesetzt werden, als auch die Methoden, die für die Verbindungen zu DsNs verwendet werden, zur Laufzeit zu beziehen und auszutauschen. Nur die Verbindung zum Basisnetz und

die darin verwendeten Protokolle müssen zur Konfigurationszeit vorgegeben werden. Durch die Flexibilität kann der Ansatz zukünftige Entwicklungen im Internet bzw. dem Kommunikationsnetz, wie z.B. neue Kommunikationsprotokolle, unterstützen.

Gliederung

Dieses Kapitel gliedert sich wie folgt. In Abschnitt 3.1 wird näher auf die Problemstellung eingegangen. Dabei wird gezeigt, dass ein flexibles Zugriffsverfahren für DsNs auf Endsystemen benötigt wird. In Abschnitt 3.2 werden relevante Arbeiten vorgestellt. In Abschnitt 3.3 wird eine Übersicht über das Zugriffsverfahren EUNA gegeben. In den darauf folgenden Abschnitten wird näher auf EUNA eingegangen. In Abschnitt 3.4 wird gezeigt, wie EUNA DsNs für einen Dienst findet und auswählt. In Abschnitt 3.5 wird genauer auf das Beziehen Dienst-spezifischer Protokolle eingegangen. In Abschnitt 3.6 wird das Aufbauen Virtueller Verbindungen zu DsNs näher behandelt. In Abschnitt 3.7 wird die Implementierung des Zugriffsverfahrens als Prototyp vorgestellt. In Abschnitt 3.8 wird EUNA anhand des Prototypen evaluiert. In Abschnitt 3.9 werden die Ergebnisse dieses Kapitels zusammengefasst.

3.1 Problemstellung

Für eine flexible Nutzung von DsNs wird ein Zugriffsverfahren benötigt, das erlaubt, DsNs für Dienste zu finden, die benötigten Protokolle für die Nutzung von DsNs zu beziehen und Verbindungen zu DsNs aufzubauen, wenn Nutzer auf Dienste zugreifen. Diese Problemstellung wird anhand des Beispiels in Abbildung 3.1 verdeutlicht.

In Abbildung 3.1 wird der Zugriff auf ein DsN schematisch dargestellt. Ein Nutzer auf einem Endsystem greift über eine Anwendung (*Ax*) auf einen Dienst zu. Das Endsystem verwendet Dienst-spezifische Protokolle, um auf den Dienst innerhalb des DsN zuzugreifen. Die Dienst-spezifischen Protokolle sind in einem Dienst-spezifischen Protokoll-Stapel (*DPS1*) enthalten und werden über eine Virtuelle Verbindung betrieben. Die Virtuelle Verbindung wird vom Endsystem zu einem virtuelle Knoten im DsN aufgebaut. Die Virtuelle Verbindung erfolgt über die Infrastruktur-Verbindungen und die Infrastruktur-Protokolle sowohl des Endsystems als auch der beteiligten Infrastruktur-Knoten (*IK*). Hieraus ergeben sich beim Zugriff auf DsNs die Teilprobleme, die im Folgenden näher beschrieben werden.

Finden und Wahl Dienst-spezifischer Netze

Ein Dienst kann in einem oder mehreren DsNs angeboten werden. Greift ein Nutzer über eine Anwendung auf einen Dienst zu, muss das Endsystem die DsNs finden, die den

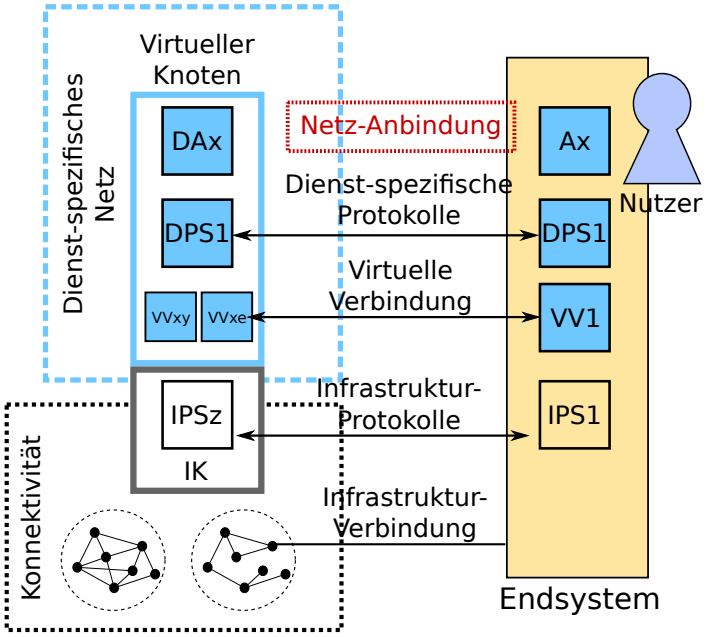

Abbildung 3.1: *Zugriff auf Dienst-spezifische Netze*

Dienst anbieten. Dies setzt außerdem voraus, dass Dienste, auf die ein Nutzer zugreift, durch das Endsystem identifiziert werden können. Wenn mehrere DsNs für einen Dienst zur Verfügung stehen, muss durch das Endsystem entschieden werden, zu welchen Netzen eine Verbindung aufgebaut werden soll. Hierfür muss eine Unterscheidbarkeit der DsNs ermöglicht werden und es müssen Entscheidungskriterien für die Wahl eines Netzes geliefert werden.

Beziehen der benötigten Dienst-spezifischen Protokolle

Jedes DsN verwendet individuelle Dienst-spezifische Protokolle, die vom Dienste-Anbieter festgelegt werden. Ist entschieden, dass eine Verbindung zu einem DsN aufgebaut werden soll, muss daher das Endsystem die Protokolle ermitteln, die innerhalb des DsN verwendet werden. Es werden also Mechanismen benötigt, die ein DsN auf eine Menge von Protokollen abbilden. Zudem werden Mechanismen benötigt, die das Beziehen der Dienst-spezifischen Protokolle ermöglichen, die innerhalb eines DsN verwendet werden. Hat ein Endsystem die Protokolle bezogen, die für die Kommunikation über das DsN benötigt werden, müssen diese auf dem Endsystem geladen und ausgeführt werden. Wird eine Verbindung zu einem DsN nicht mehr benötigt, müssen auch die entsprechenden Protokolle auf dem Endsystem nicht mehr ausgeführt werden und können entfernt werden.

Verbinden zu Dienst-spezifischen Netzen

Ein Dienste-Anbieter kann beliebige Virtuelle Knoten als Zugangspunkte zu seinem DsN festlegen, mit denen sich Endsysteme verbinden können. Virtuelle Knoten des DsN können sich in unterschiedlichen Infrastruktur-Domänen befinden. Letztere können unterschiedliche Infrastruktur-Protokolle unterstützen. Ist entschieden, dass eine Verbindung zu einem DsN aufgebaut werden soll, muss daher das Endsystem die Verbindungsmethoden zu dem DsN ermitteln. Hierzu muss das Endsystem die Protokolle ermitteln, die für die Verbindung zu dem DsN verwendet werden können. Außerdem müssen für jedes Protokoll, das für eine Verbindung genutzt werden kann, die möglichen Zugangspunkte ermittelt werden. Schließlich muss das Endsystem die Verbindung zu dem DsN aufbauen. Hierfür muss das Endsystem eine der Verbindungsmethoden und einen der Zugangspunkte auswählen und über die Netz-Infrastruktur mit Hilfe der verfügbaren Protokolle eine Verbindung zu dem DsN aufbauen.

3.2 Stand der Technik

Es existieren viele verschiedene Lösungen für Netz-Virtualisierung [19]. Ansätze wie zum Beispiel VINI [8], Trellis [12] und Cabo [28] behandeln das Erstellen, den Betrieb und die Verwaltung von Virtuellen Netzen. Dabei wird auch die Instantiierung von Virtuellen Routern und Switches behandelt oder das Aufbauen von Virtuellen Verbindungen zwischen diesen. Eine Herausforderung ist, wie Endsysteme sich mit Virtuellen Netzen verbinden können. Dieses Problem wurde im 4WARD Projekt aufgezeigt [86]. Auch das SpoVNet-Projekt [13] mit seinen spontan aufgebauten Overlays, die von Endsystemen für den Zugriff auf Dienste genutzt werden, weist viele Ähnlichkeiten mit DsNs auf. Dennoch existierte bisher keine Lösung, die Endsystemen erlaubt, DsNs für einen Dienst zu finden, die benötigten Dienst-spezifischen Protokolle zu beziehen und eine Virtuelle Verbindung zu dem DsN aufzubauen. NENA [74], das im Rahmen der Projekte 4WARD und G-Lab entwickelte Rahmenwerk für DsNs erlaubt zwar, Verbindungen zu DsNs zu verwenden und verschiedene Dienst-spezifische Protokolle zu betreiben. Wie diese Verbindungen aufgebaut und die Dienst-spezifischen Protokolle bezogen werden, wurde erst im Rahmen dieser Arbeit gelöst.

Da die drei wichtigsten Problemstellungen des Zugriffs auf DsNs das Finden von DsNs, das Beziehen der Dienst-spezifischen Protokolle und das Aufbauen Virtueller Verbindungen zu DsNs sind, wird im Folgenden näher auf relevante Arbeiten in diesen Bereichen eingegangen. Außerdem ist die Konnektivität zwischen Endsystemen, den DsNs und Komponenten des Zugriffsverfahrens ein wichtiger Aspekt. Darüber hinaus sind eine geeignete Anwendungsschnittstelle und ein Rahmenwerk für DsNs wichtige

Komponenten. Daher wird am Ende dieses Abschnittes näher auf diese beiden Aspekte eingegangen.

3.2.1 Virtuelle Verbindungen

Es existieren verschiedene Methoden, mit denen Virtuelle Verbindungen zu DsNs aufgebaut werden können. Methoden, die auf existierenden Protokollen der ISO/OSI-Schicht 4 aufbauen, sind beispielsweise VPN-Lösungen, die Tunnel über die Protokolle UDP oder TCP ermöglichen, wie OpenVPN [84] und TINC [109]. Weitere Beispiele sind Tunnel-Protokolle auf Basis von UDP wie L2TP [110] und VXLAN [66]. Andere Beispiele setzen wiederum auf andere Protokolle auf wie z.B. Tunnel über das SSH-Protokoll [129] oder Tunnel über das HTTP-Protokoll [30], die wiederum auf TCP aufsetzen. Methoden, die auf existierenden Protokollen der ISO/OSI-Schicht 3 aufbauen, sind Tunnel-Verfahren wie IP in IP Tunnel [87], GRE [27], EtherIP [44], IPSec [58] und NVGRE [105]. Darüber hinaus sind auch Verfahren, die auf existierenden Protokollen der ISO/OSI-Schicht 3 aufbauen, möglich. Ein sehr weit verbreitetes Ethernet-basiertes Virtualisierungsverfahren ist VLAN [48].

Darüber hinaus kann es wünschenswert sein, Ressourcen in der Netz-Infrastruktur für Virtuelle Verbindungen zu reservieren. Dies ist mit Reservierungsprotokollen möglich. Reservierung von Ressourcen ist beispielsweise mit dem Resource Reservation Protocol, RSVP [14], oder Next Steps in Signalling, NSIS [39], möglich.

Denkbar sind auch Methoden, die mit dem Netz kooperieren, um Virtuelle Verbindungen aufzubauen. SDN-basierte Ansätze wie “Towards Software-Friendly Networks” [128] oder “Participatory Networking: An API for Application Control of SDNs” [29] erlauben über separate Schnittstellen, dem Netz einen Kommunikationswunsch zu signalisieren und z.B. Ressourcen zu reservieren. Außerdem kann das Netz z.B. Hinweise darüber geben, wann eine Kommunikation am ehesten die geforderten Anforderungen erfüllt. Dies könnte dazu verwendet werden, dem Netz zu signalisieren, dass eine Virtuelle Verbindung zu einem DsN aufgebaut werden soll. Das Netz kann dann z.B. Ressourcen reservieren, geeignete Verbindungsmethoden ermitteln und dem Endsystem mitteilen, welches Format die Pakete haben, die mit dem DsN ausgetauscht werden.

Heutzutage ist also schon eine Vielzahl verschiedener Methoden denkbar, die von Endsystemen verwendet werden, um sich mit DsNs zu verbinden. Es ist zu erwarten, dass in Zukunft noch mehr Methoden hinzukommen. Ein Dienste-Anbieter sollte die Methoden für sein DsN selbst festlegen, da nur dieser entscheiden kann, welche Methoden am sinnvollsten für seinen Dienst sind. Dabei sollte dem Dienste-Anbieter erlaubt werden, beliebige Methoden für Virtuelle Verbindungen zu verwenden. Daher müssen auch Endsysteme in der Lage sein, verschiedene Methoden zu verwenden, und benötigen Wissen darüber, welche Methoden für ein DsN zur Verfügung stehen. Außerdem müssen die Methoden für Virtuelle Verbindungen parametrisiert bzw. konfiguriert werden.

Zum Beispiel bei den UDP-basierten Ansätzen wird das Wissen benötigt, zu welchen IP-Adressen und UDP-Ports eine Verbindung aufgebaut werden kann. Für Nutzerfreundlichkeit sollte der Nutzer weitestgehend nicht im Aufbau der Virtuellen Verbindungen involviert werden. Bisher existierte kein Verfahren, das alle diese Anforderungen erfüllt. Das in diesem Kapitel vorgestellte Zugriffsverfahren erfüllt diese Anforderungen und erlaubt, Virtuelle Verbindungen mit Hilfe beliebiger Methoden aufzubauen, ohne den Nutzer dabei einzubeziehen.

3.2.2 Beziehen der Protokolle

Es existieren verschiedene Methoden, um die Dienst-spezifischen Protokolle eines DsN zu beziehen. Software-Verteil-Systeme sind eine weit verbreitete Lösung, um Software Updates oder Software Pakete zwischen Endsystemen und so genannten Repositories auszutauschen. Das Advanced Packaging Tool (APT) [114] ist ein sehr verbreitetes System, das in Debian Linux und seinen Derivaten eingesetzt wird, um auf Endsystemen Software Pakete zu installieren, zu aktualisieren und zu entfernen. APT verwendet zentralisierte Repositories, auf denen sowohl der Katalog verfügbarer Software-Pakete als auch die Software Pakete selbst gespeichert werden. Bei Repositories handelt es sich um Webserver oder FTP-Server. Um ein bestimmtes Software-Paket zu erhalten, gibt der Nutzer den Namen des Paketes an und APT lädt das Paket von einem Repository herunter und installiert es zusammen mit anderen Software-Paketen, die für die Nutzung des gewünschten Software-Paketes benötigt werden (so genannte Abhängigkeiten). Ein Software Paket ist mit Hilfe der Signatur der für das Paket verantwortlichen Person (so genannter Maintainer) gesichert. Die öffentlichen Schlüssel der Maintainer werden in einem Schlüssel-Bund auf einem Repository gespeichert. Hierdurch muss der Nutzer dem Repository vertrauen. Ist ein Repository kompromittiert, kann ein Angreifer die Pakete, die Signaturen, die öffentlichen Schlüssel oder die Liste der Abhängigkeiten ändern. Außerdem muss der Nutzer den Maintainern der Software Pakete vertrauen. Je nach Abhängigkeiten eines Paketes muss der Nutzer einer Kette von Maintainern vertrauen. Wenn ein Paket beispielsweise drei andere Pakete benötigt, muss der Nutzer dem Paket Maintainer, dem Repository und drei anderen Maintainern vertrauen.

Wie bei dem vorgestellten APT bieten sich also für das Beziehen von Dienst-spezifischen Protokollen Verfahren an, die auf dem Client/Server Prinzip basieren. Verbreitet sind Webserver und das HTTP-Protokoll [30] sowie FTP-Server und das FTP-Protokoll [91]. Bei diesen Ansätzen werden die Protokolle auf zentralisierten Servern abgelegt und von Endsystemen heruntergeladen. Bei einer großen Anzahl von Endsystemen kann es bei einem solchen Verfahren zu Engpässen auf den zentralisierten Servern kommen. Um die Skalierbarkeit zu erhöhen, bieten sich Peer-to-Peer Systeme wie das Protokoll Bittorrent [20] an. Hierbei werden Dienst-spezifische Protokolle nicht von einem zentralisierten Server heruntergeladen. Stattdessen kann ein Endsystem Dienst-spezifische

Protokolle von anderen Endsystemen beziehen, die das entsprechende Dienst-spezifische Protokoll lokal gespeichert haben.

Die Verwendung von DsNs erlaubt dem Dienste-Anbieter, beliebige Dienst-spezifische Protokolle in seinem DsN zu betreiben. Daher benötigt ein Endsystem beim Beziehen der Protokolle Wissen darüber, welche Dienst-spezifischen Protokolle für ein DsN bezogen werden müssen. Außerdem sollte der Dienste-Anbieter entscheiden können, wie die Dienst-spezifischen Protokolle publiziert bzw. von Endsystemen bezogen werden können, um Flexibilität zu erreichen. Hat ein Dienste-Anbieter Zugriff auf HTTP-Server, soll er diese verwenden können. Erwartet ein Dienste-Anbieter eine große Anzahl von Endsystemen oder steht ihm keine eigene Infrastruktur zur Verfügung, soll das Beziehen der Protokolle auch über Peer-to-Peer Systeme möglich sein. Daher benötigt das Endsystem Wissen darüber, wie die Dienst-spezifischen Protokolle bezogen werden können. Auch die Parameter und Konfiguration für eine vom Dienste-Anbieter festgelegte Bezugsmethode muss dem Endsystem verfügbar gemacht werden. Das Beziehen von Dienst-spezifischen Protokollen sollte, soweit möglich, ohne Interaktion mit Nutzern geschehen, da der Nutzer dies als störend empfinden könnte. Außerdem sollte der Nutzer beim Beziehen der Dienst-spezifischen Protokolle nur dem Dienste-Anbieter des entsprechenden DsN vertrauen müssen. Bisher existierte kein Verfahren, das alle diese Anforderungen erfüllt. Das in diesem Kapitel vorgestellte Zugriffsverfahren erfüllt diese Anforderungen. Es bietet die Flexibilität, beliebige Methoden zu verwenden, und erlaubt das Beziehen von Dienst-spezifischen Protokollen, ohne den Nutzer dabei einbeziehen zu müssen.

3.2.3 Finden Dienst-spezifischer Netze

Das Finden von DsNs für einen Dienst ist vergleichbar mit dem Finden von Diensten bzw. Dienst-Instanzen, das durch so genannte Service Discovery Protokolle erreicht werden kann. In lokalen Netzen erlaubt das Protokoll DHCP [24], das verwendet wird, um Endsysteme in einem lokalen Netz dynamisch zu konfigurieren, dem Endsystem Dienste des Netzes, wie z.B. zur Namensauflösung, zum Drucken oder zum Einstellen der Uhrzeit, mitzuteilen. Multicast-DNS [18], eine Multicast-basierte Erweiterung von DNS [80], erlaubt DNS-Abfragen in lokalen Netzen ohne zentrale Infrastruktur. Dies kann in Verbindung mit DNS-Based Service Discovery [17] verwendet werden, um in SRV-, TXT- und PTR-Einträgen im DNS Dienst-Instanzen in lokalen Netzen zu publizieren bzw. anzukündigen. Das Service Location Protocol (SLP) [37] verwendet auch Multicast-Kommunikation und die Protokolle UDP und TCP, um Dienste in einem lokalen Netz zu finden. Das Suchen von Diensten verläuft dabei über Multicast. SLP führt zusätzliche optionale Infrastruktur ein, die sogenannten Directory Agents. Directory Agents speichern angekündigte Dienste und werden verwendet, um Skalierbarkeit in großen Netzen zu verbessern.

Im Kontext von Service Oriented Architectures (SOA) und Web Services spielt auch das Finden von Diensten eine wichtige Rolle. Ein Multicast-basiertes Verfahren für lokale Netze ist das Protokoll Web Services Dynamic Discovery, WS-Discovery [116]. Es nutzt die Protokolle IP, UDP und TCP, um Web Services in einem lokalen Netz zu finden. Ein weiteres Verfahren ist Universal Description, Discovery and Integration, UDDI [112]. Es benutzt so genannte Registries (White, Yellow und Green Pages), die von Endsystemen abgefragt werden, um Dienste zu finden und Informationen darüber zu erhalten, wie auf die Dienste zugegriffen werden kann. Mit Hilfe der Registries ist auch ein Finden von Diensten im Internet, also über lokale Netze hinaus, möglich.

Um Dienste über lokale Netze hinaus anzubieten bzw. zu finden und die Skalierbarkeit zu erhöhen, bieten sich Ansätze wie das zuvor erwähnte DNS-Based Service Discovery an. Um eine hohe Skalierbarkeit zu erreichen, bietet sich an, Informationen über Dienste in Verteilten Hash-Tabellen (Distributed Hash Table, DHT) wie zum Beispiel Chord [107] oder Kademlia [76] zu speichern. DHT-basierte Ansätze werden beispielsweise verwendet, um die Skalierbarkeit der zuvor vorgestellten Verfahren WS-Discovery und UDDI zu erhöhen. In [2] wird ein Ansatz vorgestellt, der WS-Discovery mit einer DHT kombiniert. Hierdurch erreicht der Ansatz auch das Finden über lokale Netze hinausgehend. In [7] wird ein Ansatz vorgestellt der UDDI mit einer DHT kombiniert. Auch in SpoVNet [13] wird vorgeschlagen, das Finden von Overlay-Netzen für SpoVNet-Dienste über eine DHT im Basis-Overlay durchzuführen. Allerdings wurde hierfür keine konkrete Lösung vorgestellt.

Die hier vorgestellten Ansätze zeigen, dass eine Vielzahl verschiedener Möglichkeiten existieren, Dienste bzw. Dienst-Instanzen zu finden. Anders als beim Aufbauen Virtueller Verbindungen und dem Beziehen von Protokollen, können zum Finden von DsNs allerdings nicht beliebige Protokolle verwendet werden. Das Finden von DsNs stellt sozusagen den Einstiegspunkt in das Zugriffsverfahren dar und muss für alle Endsysteme möglich sein. Daher verwendet das Zugriffsverfahren für DsNs einen Mapping-Dienst in einem Basisnetz. In der Implementierung des Mapping-Dienstes wird eine DHT verwendet. Allerdings ist das Zugriffsverfahren nicht auf die Verwendung einer DHT eingeschränkt. Um Änderungen in Zukunft zu erlauben, ist das Zugriffsverfahren modular aufgebaut, was eine Änderung der Protokolle für den Zugriff auf den Mapping-Dienst oder eine Änderung des Basisnetzes vereinfacht.

3.2.4 Konnektivität

Ein wichtiger Aspekt bei DsNs ist die grundlegende Konnektivität zwischen Endsystemen, DsNs und den Komponenten des Zugriffsverfahrens. Endsysteme benötigen eine solche grundlegende Konnektivität, um DsNs finden, Dienst-spezifische Protokolle beziehen und Virtuelle Verbindungen zu DsNs aufbauen zu können. In Genesis [61] und Tempest [77] wird vorgeschlagen, grundlegende Konnektivität über ein gemeinsames Basisnetz zu

erreichen. In diesen Ansätzen wird dieses Basisnetz für die Signalisierung zwischen Dienste-Anbietern bzw. Betreibern Virtueller Netze und Infrastruktur-Betreibern und für das Bootstrapping neuer Virtueller Netze verwendet. Endsysteme werden nicht berücksichtigt. SpoVNet [13] verwendet ein Basisnetz, das Basis-Overlay, für eine grundlegende Konnektivität zwischen Endsystemen. Hierbei wird auch eine Konnektivität über eine heterogene Netz-Infrastruktur mit so genannten Connectivity Domains [78] erreicht. Das in diesem Kapitel vorgestellte Zugriffsverfahren setzt ebenfalls auf ein Basisnetz, um eine grundlegende Konnektivität zwischen Endsystemen, DsNs und den Komponenten des Zugriffsverfahrens zu erreichen.

3.2.5 Anwendungsschnittstelle und Rahmenwerk

Ein weiterer wichtiger Aspekt einer Lösung für den Zugriff auf DsNs ist die Anwendungsschnittstelle, die Anwendungen nutzen, um auf Dienste bzw. DsNs zuzugreifen. Beispielsweise wird in Name-based Sockets [111] vorgeschlagen, die existierende Socket-API in sofern zu erweitern, dass sie abstrakte Namen verwendet. Hierdurch werden Namensauflösung und Protokoll-Selektion unter die Anwendungsschnittstelle und in den Netzwerk-Stapel verschoben. Allerdings berücksichtigt dies nicht die Abbildung der abstrakten Namen auf (Dienst-spezifische) Netze, die Auswahl von Netzen oder die Abbildung von Netzen auf Protokolle, die in den Netzen verwendet werden. Die NENA-API [71] wurde für DsNs entworfen und erlaubt, diese Schritte unterhalb der Schnittstelle umzusetzen. Daher wurde die NENA-API für die Realisierung des in diesem Kapitel vorgestellten Zugriffsverfahren für DsNs verwendet.

Für den Zugriff auf DsNs wird außerdem ein Rahmenwerk benötigt, das es erlaubt Dienst-spezifische Protokolle zu laden und auszuführen. Außerdem muss das Rahmenwerk in der Lage sein, Virtuelle Verbindungen zu DsNs zu verwenden. Hierfür muss das Rahmenwerk die Dienst-spezifischen Protokolle den Virtuellen Verbindungen zuordnen können. NENA [74], das Rahmenwerk für DsNs, wurde im Rahmen dieser Arbeit um die geforderten Eigenschaften erweitert und für die Realisierung und Evaluierung des in diesem Kapitel vorgestellten Zugriffsverfahrens verwendet.

3.3 Flexibles Zugriffsverfahren für Dienst-spezifische Netze

In diesem Abschnitt wird *EUNA* (*End User Node Access*), das flexible Zugriffsverfahren für DsNs, vorgestellt. Dabei wird ein Überblick über die Komponenten und den Ablauf des Zugriffsverfahrens gegeben. Wenn ein Nutzer auf einen Dienst zugreift, findet EUNA DsNs, die diesen Dienst anbieten. EUNA wählt eines dieser DsNs aus, bezieht die im DsN verwendeten Dienst-spezifischen Protokolle und baut eine Virtuelle Verbindung zu dem DsN auf. EUNA erlaubt das Austauschen der Methoden zum Beziehen der Protokolle

und das Austauschen der Methoden für das Etablieren der Virtuellen Verbindungen. Der Betreiber eines DsN besitzt die Kontrolle darüber, welche Dienst-spezifischen Protokolle, welche Bezugsmethoden und welche Verbindungsmethoden für sein DsN verwendet werden. Das Endsystem kann aus den Vorgaben des Dienste-Anbieters geeignete Protokolle auswählen. Außerdem erlaubt EUNA dem Endsystem, den Zugriff transparent für den Nutzer durchzuführen.

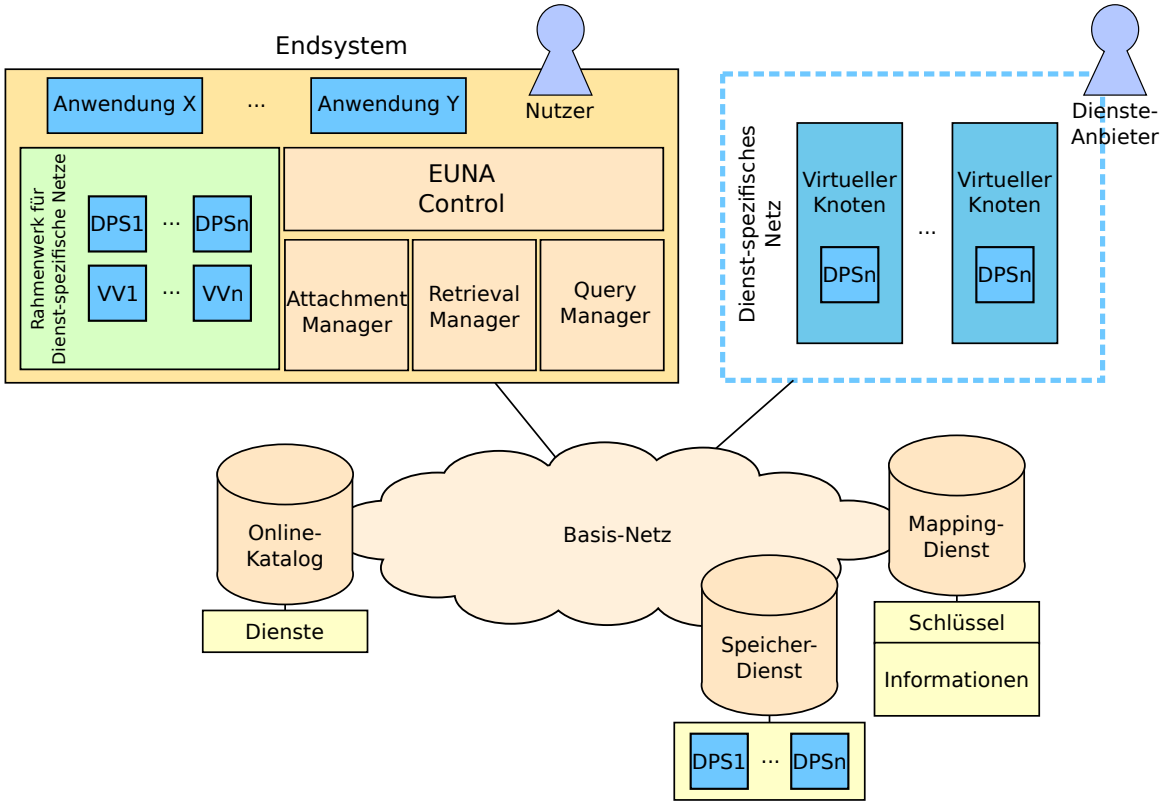

Abbildung 3.2: *Übersicht über Komponenten des Zugriffsverfahrens für Dienst-spezifische Netze*

Eine Übersicht über die Komponenten von EUNA wird in Abbildung 3.2 dargestellt. In der Abbildung werden beispielhaft ein Endsystem, ein DsN und die Netzkomponenten für das Zugriffsverfahren dargestellt. Die Netzkomponenten sind ein Online-Katalog, ein Speicher-Dienst, ein Mapping-Dienst und das Basisnetz. Sowohl das Endsystem als auch das DsN sind mit dem Basisnetz verbunden. Die Komponenten auf einem Endsystem sind ein Rahmenwerk für DsNs, *EUNA Control*, *Query Manager*, *Retrieval Manager* und *Attachment Manager*. Darüber hinaus wird ein Namensschema für Dienste verwendet (nicht in der Abbildung). Die Komponenten werden im Folgenden näher vorgestellt.

3.3.1 Komponenten

EUNA verwendet ein Namensschema für Dienste und Netzkomponenten. Diese sind ein Online-Katalog, ein Mapping-Dienst, ein Speicher-Dienst und ein Basisnetz.

3.3.1.1 Namensschema

Um Dienste auf DsNs abbilden zu können, definiert EUNA ein Namensschema für Dienste. Dienste werden über URIs [11] identifiziert. Die für das Zugriffsverfahren verwendeten URIs bestehen aus folgendem Format:

```
<Dienst>:<Dienst-spezifischer Teil>
```

Die Namen sind getrennt in *Dienst* und *Dienst-spezifischer* Teil. *Dienst* gibt den Namen des Dienstes an und legt die Syntax und Semantik des Dienst-spezifischen Teils fest. *Dienst* wird dazu benutzt, um DsNs zu finden, die den entsprechenden Dienst anbieten. Beispiele für Namen sind:

```
(1) videoStreaming://onDemand/movieXYZ
(2) banking://bankX/AccountY
(3) ccn://contentname
(4) gameXYZ://server23
```

Im ersten Beispiel identifiziert der Name einen Film, der über ein Video Streaming Netz per Video On Demand bezogen werden kann. Über den Namen im zweiten Beispiel kann ein Nutzer auf ein Bankkonto bei einer Bank in einem Online-Banking Netz zugreifen. In (3) wird ein Inhalt in einem Content Centric Network (CCN) adressiert. In (4) wird ein Server für ein Online Spiel angegeben.

3.3.1.2 Online-Katalog

Um Dienste finden zu können, wird ein Online-Katalog verwendet. Der Online-Katalog ist vergleichbar mit dem heutigen World Wide Web. Über ihn wird das Auffinden von Diensten über Querverweise oder Suchfunktionen ermöglicht. Auf ihn kann der Nutzer mit Hilfe einer Anwendung vergleichbar mit einem heutigen Webbrowser zugreifen. Die über den Online-Katalog gefundenen Dienste werden über Namen identifiziert, die dem zuvor beschriebenen Namensschema entsprechen. Der Online-Katalog ist über das Basisnetz erreichbar.

3.3.1.3 Speicher-Dienst

Um das Beziehen von Dienst-spezifischen Protokollen zu ermöglichen, benutzt EUNA einen Speicher-Dienst. Im Speicher-Dienst können Dienst-spezifische Protokolle

von Dienste-Anbietern abgelegt und von Endsystemen abgerufen werden. Ein Dienste-Anbieter legt für sein DsN fest, wie die Dienst-spezifischen Protokolle aus dem Speicher-Dienst bezogen werden können. Dabei bestimmt der Dienste-Anbieter sowohl die Protokolle für das Beziehen als auch die Orte, von denen die Dienst-spezifischen Protokolle bezogen werden. Dies erlaubt beispielsweise eine Realisierung des Speicher-Dienstes mit Hilfe von Peer-to-Peer Protokollen wie Bittorrent oder basierend auf Client/Server-Protokollen wie HTTP. Um einen grundlegenden Speicher-Dienst zu bieten, sieht EUNA eine Realisierung des Speicher-Dienstes im Basisnetz vor. Der Speicher-Dienst besteht dabei aus so genannten Repositories. Die Repositories sind öffentliche Datenspeicher im Basisnetz, die zum Ablegen und Abrufen der Dienst-spezifischen Protokolle verwendet werden.

3.3.1.4 Mapping-Dienst

EUNA verwendet einen Mapping-Dienst im Basisnetz, um (1) Dienste auf DsNs (2) DsNs auf Dienst-spezifische Protokolle, (3) Dienst-spezifische Protokolle auf deren Bezugsmethoden, (4) DsNs auf deren öffentlichen Schlüssel und (5) DsNs auf Verbindungsmethoden abzubilden. Im Mapping-Dienst werden hierfür entsprechende Informationen durch die Dienste-Anbieter abgelegt und durch die Endsysteme abgerufen.

(1) Dienst → DsNs Für das Finden von DsNs für einen Dienst wird eine Abbildung des Dienstes auf eine Menge verfügbarer DsNs und dazugehöriger Metadaten im Mapping-Dienst abgelegt. Der Dienst entspricht dabei dem Namen des Dienstes, wie er im Namensschema (s.o.) festgelegt worden ist. Beispiele für Metadaten sind Kosten, Quality-of-Service-, Quality-of-Experience-Eigenschaften und Service Level Agreements.

(2) DsN → Dienst-spezifische Protokolle Um zu ermitteln, welche Dienst-spezifischen Protokolle in einem DsN verwendet werden, wird im Mapping-Dienst eine Abbildung von dem DsN auf die darin verwendeten Protokolle abgelegt.

(3) Dienst-spezifisches Protokoll → Bezugsmethoden, Bezugspunkte Für das Beziehen Dienst-spezifischer Protokolle wird für jedes Dienst-spezifische Protokoll eines DsN eine Abbildung auf dessen, vom Dienste-Anbieter verwendete, Bezugsmethoden gespeichert. Die Bezugsmethoden geben Protokolle wie zum Beispiel HTTP, FTP, Bittorrent oder CCN [55] an. Die Bezugsmethoden können in eigenen DsNs oder dem Basisnetz verwendet werden. Daher gibt die Bezugsmethode auch an, in welchem Netz sie verwendet wird. Für jede Bezugsmethode werden außerdem Bezugspunkte angegeben, von denen die Dienst-spezifischen Protokolle bezogen werden können. Die Bezugspunkte sind abhängig von der entsprechenden Bezugsmethode: Für Bezugsmethoden wie HTTP und FTP können URLs oder Kombinationen aus Server-Adressen, Datei-Pfaden und Dateinamen verwendet werden. Für Peer-to-Peer basierte Bezugsmethoden wie Bittorrent können die

Bezugspunkte die Identifikationsnummern von Dienst-spezifischen Protokollen und von anderen Endsystemen sein. Für Bezugsmethoden wie CCN kann beispielsweise ein eindeutiger Name des Dienst-spezifischen Protokolls benutzt werden. Damit ein Endsystem Bezugsmethoden und Bezugspunkte auswählen kann, werden zusätzlich Selektionsmethoden angegeben. Beispiele für Selektionsmethoden sind eine zufallsgesteuerte Wahl oder eine Wahl basierend auf der Reihenfolge der Bezugsmethoden bzw. Bezugspunkte.

(4) DsN → öffentlicher Schlüssel Um zu verhindern, dass fehlerhafte oder manipulierte Dienst-spezifische Protokolle auf einem Endsystem ausgeführt werden, überprüft EUNA die Korrektheit bezogener Protokolle mit Hilfe öffentlicher Schlüssel (genauer: kryptographischer Signaturen). Damit ein Endsystem den öffentlichen Schlüssel eines DsN erhält, wird eine Abbildung auf den öffentlichen Schlüssel des DsN im Mapping-Dienst abgelegt.

(5) DsN → Verbindungsmethoden, Zugangspunkte Für das Etablieren einer Virtuellen Verbindung zu einem DsN wird im Mapping-Dienst eine Abbildung von dem DsN auf dazugehörige Verbindungsmethoden abgelegt. Die Verbindungsmethoden geben Protokolle an, mit denen Verbindungen zu dem DsN aufgebaut werden können. Beispiele hierfür sind UDP-Tunnel oder Ethernet mit VLANs. Die Verbindungsmethoden geben außerdem an, ob sie über die Netz-Infrastruktur bzw. bestimmte Infrastruktur-Domänen, DsNs oder das Basisnetz betrieben werden. Darüber hinaus wird für jede Verbindungsmethode eine Menge von Zugangspunkten angegeben, zu denen Virtuelle Verbindungen aufgebaut werden können. Für Verbindungsmethoden wie UDP-Tunnel kann ein Zugangspunkt beispielsweise eine IP-Adresse und ein UDP-Port sein. Für eine Verbindung über ein VLAN kann eine VLAN-ID angegeben werden. Damit ein Endsystem Verbindungsmethoden und Zugangspunkte auswählen kann, werden zusätzlich Selektionsmethoden angegeben. Beispiele für Selektionsmethoden sind eine zufallsgesteuerte Wahl oder eine Wahl basierend auf der Reihenfolge der Verbindungsmethoden bzw. Zugangspunkte.

3.3.1.5 Basisnetz

Die Konnektivität zwischen den Endsystemen, DsNs und den verschiedenen Diensten wird über ein Basisnetz sichergestellt. Das Basisnetz ist ein spezielles DsN, zu dem alle Endsysteme verbunden sind. Endsysteme greifen darüber auf den Mapping-Dienst zu. Zudem werden darin Repositories für den Speicher-Dienst betrieben. Außerdem erlaubt es Endsystemen das Etablieren von Virtuellen Verbindungen zu DsNs, falls keine direkte Konnektivität über andere DsNs oder die Netz-Infrastruktur möglich ist. Die Protokolle, die innerhalb des Basisnetzes verwendet werden, werden von Endsystemen offline bezogen.

3.3.2 Endsysteme

Der Aufbau eines Endsystems wird in Abbildung 3.3 dargestellt. Es besteht aus einer Anwendungsschicht, einem Rahmenwerk für DsNs, der Virtualisierungsschicht und der Netz-Infrastruktur-Schicht. Diese Schichten werden im Folgenden näher beschrieben.

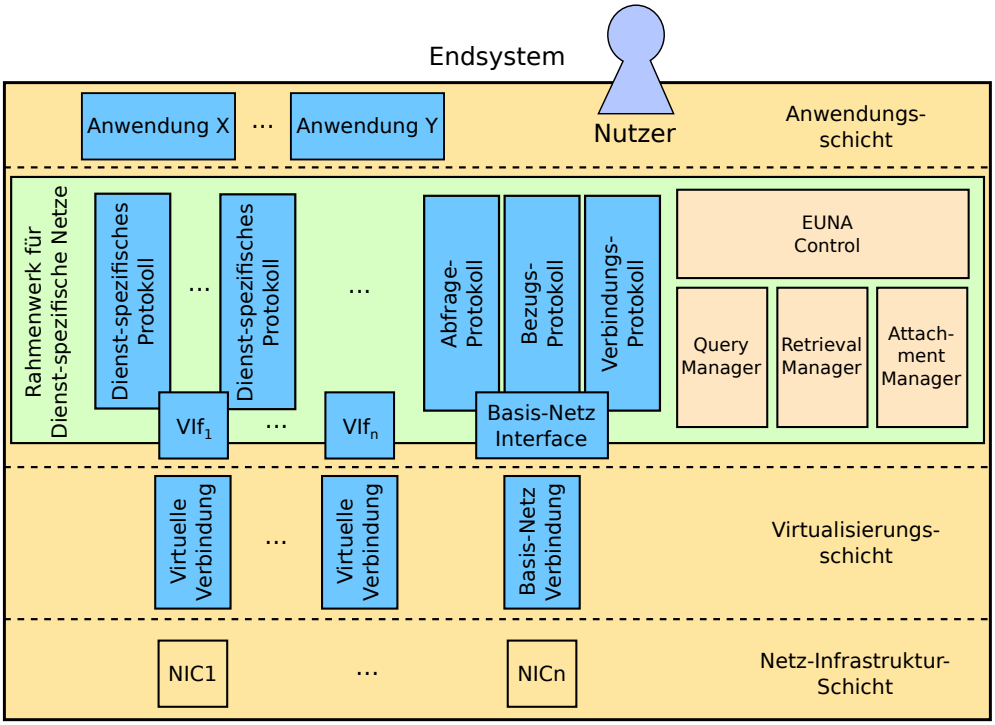

Abbildung 3.3: *Aufbau von Endsystemen*

3.3.2.1 Anwendungsschicht

In der Anwendungsschicht befinden sich die Anwendungen des Nutzers. Die Anwendungen greifen über eine abstrakte Anwendungsschnittstelle, die Netz-Details vor der Anwendung verbirgt, auf das Rahmenwerk zu. Als Beispiel wird im folgenden Pseudo-Code eine Video-Streaming Anwendung gezeigt, die einen Video-Streaming Dienst in einem DsN nutzt, um ein Video zu empfangen und abzuspielen.

```
handle = get(VideoURI)
while not handle.endOfStream():
    videoData = handle.read()
    play(videoData)
```

Die Anwendung kommuniziert über die Anwendungsschnittstelle mit dem Rahmenwerk. Der Befehl *get(VideoURI)* initiiert die Kommunikation. Dieser Aufruf blockiert die Anwendung bis das Rahmenwerk das Handle (*handle*) zurückliefert. Über das Handle

werden Daten mit dem Dienst ausgetauscht. In einer Schleife liest die Anwendung mit dem Befehl *handle.read()* Video-Daten vom Handle und spielt diese über *play(videoData)* ab.

3.3.2.2 Rahmenwerk für Dienst-spezifische Netze

Auf dem Endsystem wird ein Rahmenwerk wie NENA [74] verwendet. Das Rahmenwerk ermöglicht die Verwendung zur Laufzeit aufgebauter Virtueller Verbindungen zu DsNs sowie das Nachladen und Ausführen von Protokollen. Es greift über virtuelle Netzwerkschnittstellen auf die Virtualisierungsschicht zu und enthält die EUNA-Komponenten *EUNA Control*, *Query Manager*, *Retrieval Manager* und *Attachment Manager*. Über eine virtuelle Netzwerkschnittstelle greift es auf das Basisnetz zu, zu dem eine Verbindung beim Start des Rahmenwerkes aufgebaut wird. Für das Basisnetz werden Protokolle zum Abrufen von Informationen aus dem Mapping-Dienst, Abrufen von Dienst-spezifischen Protokollen aus dem Speicher-Dienst und Aufbauen von Virtuellen Verbindungen betrieben.

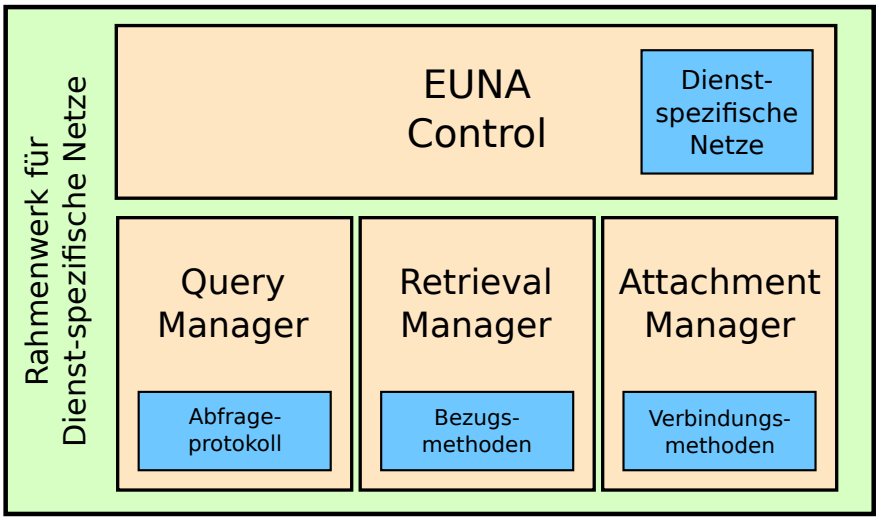

Abbildung 3.4: *Übersicht über die EUNA-Komponenten im Rahmenwerk für Dienst-spezifische Netze*

Die EUNA-Komponenten werden in Abbildung 3.4 dargestellt. Die wichtigste Komponente ist *EUNA Control*. *EUNA Control* greift auf die Komponenten *Query Manager*, *Retrieval Manager* und *Attachment Manager* zu. *EUNA Control* speichert die *Dienst-spezifischen Netze*, zu denen das Endsystem verbunden ist, und deren Dienst-Namen in einer Liste ab. Der *Query Manager* speichert eine Referenz auf das *Abfrageprotokoll* im Basisnetz. Der *Retrieval Manager* unterhält eine Liste der auf dem Endsystem verfügbaren *Bezugsmethoden*. Der *Attachment Manager* verwaltet eine Liste der auf dem Endsystem verfügbaren *Verbindungsmethoden*.

EUNA Control *EUNA Control* verwaltet das Zugriffsverfahren auf einem Endsystem. Hierfür verwendet es die anderen Komponenten, die wiederum für die einzelnen Schritte des Zugriffsverfahrens verantwortlich sind. Der *Query Manager* wird verwendet, um Informationen aus dem Mapping-Dienst abzufragen. Der *Retrieval Manager* wird verwendet, um Protokolle aus dem Speicher-Dienst zu beziehen. Der *Attachment Manager* wird verwendet, um Virtuelle Verbindungen zu DsNs aufzubauen.

Query Manager Der *Query Manager* ruft Informationen aus dem Mapping-Dienst ab und stellt sie *EUNA Control* bereit. Er ist dem *Abfrageprotokoll* des Basisnetzes im Rahmenwerk zugeordnet. Das Abfrageprotokoll realisiert die Kommunikation mit dem Mapping-Dienst.

Retrieval Manager *EUNA Control* nutzt den *Retrieval Manager*, um Protokolle für DsNs abzurufen. Der *Retrieval Manager* kann mehrere Bezugsmethoden verwenden, die die Kommunikation mit dem Speicher-Dienst realisieren. Eine Bezugsmethode wird durch das *Bezugsprotokoll* im Rahmenwerk bereit gestellt. Darüber hinaus können andere Dienst-spezifische Protokolle im Rahmenwerk als Bezugsmethoden verwendet werden.

Attachment Manager *EUNA Control* nutzt den *Attachment Manager*, um Virtuelle Verbindungen zu DsNs aufzubauen. Der *Attachment Manager* greift wiederum auf die Virtualisierungsschicht und DsNs zu, um Virtuelle Verbindungen aufzubauen. Er kann die dort verfügbaren Verbindungsmethoden verwenden. Falls ein Verbindungsaufbau über die Virtualisierungsschicht oder DsNs nicht möglich ist, kann er die Verbindung über das *Verbindungsprotokoll* im Basisnetz aufbauen.

Wenn eine Anfrage, wie zum Beispiel *get(VideoURI)*, einer Anwendung über die Anwendungsschnittstelle in dem Rahmenwerk eintrifft, wird das Zugriffsverfahren gestartet. Das Endsystem fragt die Informationen zum DsN aus dem Mapping-Dienst ab, bezieht die Dienst-spezifischen Protokolle und baut eine Virtuelle Verbindung zu dem DsN auf. Die Dienst-spezifischen Protokolle werden im Rahmenwerk geladen. Außerdem wird die Virtuelle Verbindung zum DsN im Rahmenwerk mit den Dienst-spezifischen Protokollen assoziiert. Nachdem erfolgreich auf das DsN zugegriffen worden ist, bestätigt das Rahmenwerk die Anfrage der Anwendung mit einem Handle. Die Anwendung nutzt das Handle, um Daten mit dem Dienst auszutauschen. Der weitere Datenaustausch zwischen der Anwendung und dem Dienst verläuft über das Rahmenwerk ohne weitere Interaktion mit EUNA.

3.3.2.3 Virtualisierungsschicht

Die Virtualisierungsschicht wird dafür verwendet, Virtuelle Verbindungen zu etablieren und zu verwalten. Die Virtuellen Verbindungen werden über die Netz-Infrastruktur bzw. die Netzwerkschnittstellen des Endsystems betrieben. Zum Aufbauen Virtueller

Verbindungen stellt die Virtualisierungsschicht dem *Attachment Manager* Verbindungsmethoden, wie z.B. auf dem Endsystem verfügbare Infrastruktur-Protokolle, bereit.

3.3.2.4 Netz-Infrastruktur-Schicht

In der Netz-Infrastruktur-Schicht befinden sich die Netzwerkschnittstellen des Endsystems. Über diese ist das Endsystem an die Netz-Infrastruktur bzw. an Infrastruktur-Domänen angeschlossen.

3.3.3 Ablauf

EUNA ist in zwei separate Stufen aufgeteilt. Die erste Stufe wird vom Dienste-Anbieter und die zweite Stufe vom Endsystem durchgeführt. Eine kurze Zusammenfassung des Ablaufs ist wie folgt:

(1) Der Dienste-Anbieter führt folgende Schritte zusammen mit der Erstellung eines DsN für einen Dienst durch:
   - Erstellung des DsN und der Zugangspunkte
   - Speichern der Dienst-spezifischen Protokolle im Speicher-Dienst
   - Publizieren der von Endsystemen für den Zugriff auf das DsN benötigten Informationen im Mapping-Dienst

(2) Das Endsystem führt nach den oben genannten Schritten des Dienste-Anbieters die folgenden Schritte durch, wenn ein Nutzer auf den Dienst zugreift, der in dem DsN des Dienste-Anbieters realisiert wird:
   - Abrufen der Informationen aus dem Mapping-Dienst
   - Beziehen der Protokolle aus dem Speicher-Dienst
   - Aufbauen einer Virtuellen Verbindung zum DsN
   - Nutzen des Dienstes über das DsN

Im Folgenden werden die Schritte, die auf der Seite des Dienste-Anbieters und auf der Seite des Endsystems durchgeführt werden, näher beschrieben.

3.3.3.1 Dienste-Anbieter

Im ersten Schritt spezifiziert der Dienste-Anbieter das DsN inklusive der Netzstruktur und Quality-of-Service-Parameter passend zu den Anforderungen des Dienstes, den er anbieten möchte. Die Spezifikation übermittelt der Dienste-Anbieter an den Infrastruktur-Anbieter. Der Infrastruktur-Anbieter erstellt daraufhin das DsN und übergibt sämtliche

Informationen zur Konfiguration der Virtuellen Knoten an den Dienste-Anbieter. Bei der Erstellung des DsN findet auch die Festlegung von Verbindungsmethoden und Zugangspunkten zu dem DsN statt. Darüber hinaus spezifiziert der Dienste-Anbieter einen Namen, unter dem der Dienst bekannt gegeben werden soll, und somit ein URI-Namensschema. Der Dienste-Anbieter legt außerdem die Dienst-spezifischen Protokolle fest, die für die Nutzung seines Dienstes in dem DsN verwendet werden. Der Dienste-Anbieter konfiguriert die Virtuellen Knoten, indem er die Dienst-spezifischen Protokolle auf sie lädt und entsprechend seinen Anforderungen parametrisiert. Damit die Endsysteme die Dienst-spezifischen Protokolle beziehen können, legt sie der Dienste-Anbieter im Speicher-Dienst ab. Abschließend legt der Dienste-Anbieter alle Informationen, die die Endsysteme für den Zugriff auf das DsN benötigen, als Einträge im Mapping-Dienst ab.

3.3.3.2 Endsystem

Sobald der Nutzer eines Endsystems auf den Dienst des Dienste-Anbieters zugreift und das Endsystem noch nicht mit dem entsprechenden DsN verbunden ist, wird das Zugriffsverfahren durchgeführt.

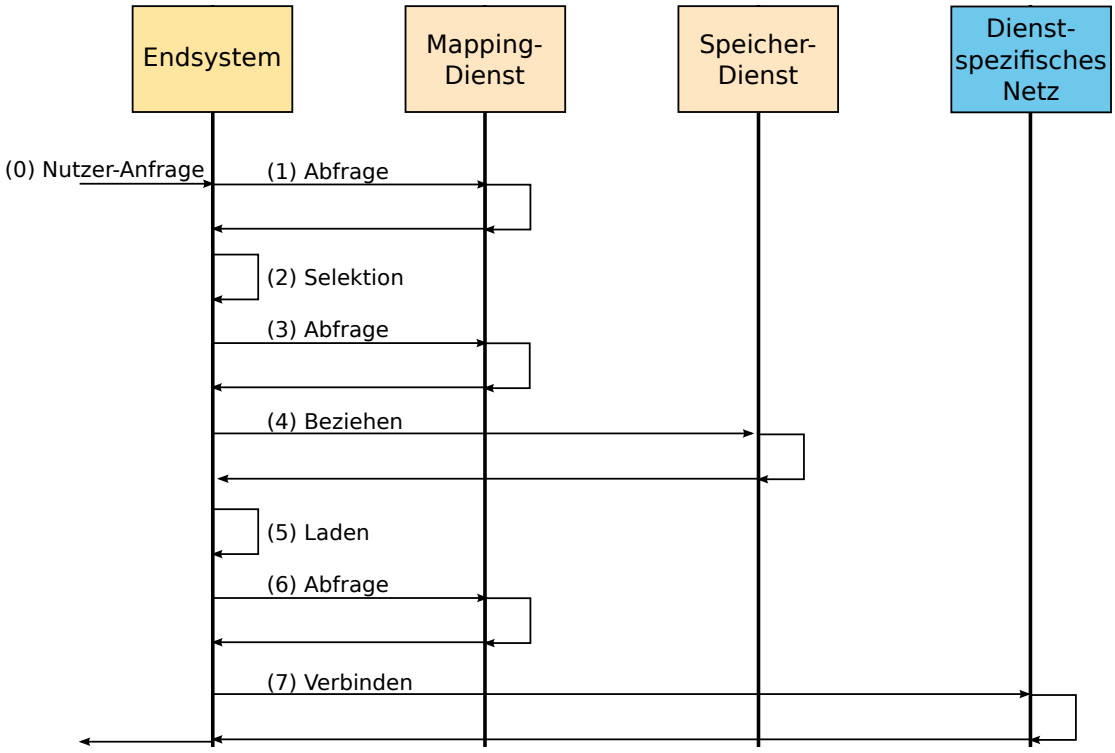

Abbildung 3.5: *Ablauf auf einem Endsystem*

In Abbildung 3.5 wird der Ablauf des Zugriffsverfahrens mit Hilfe eines Sequenzdiagramms schematisch dargestellt. Die am Zugriffsverfahren beteiligten Komponenten und

deren Interaktion werden gezeigt. Die beteiligten Komponenten sind das Endsystem (welches die zuvor beschriebenen EUNA-Komponenten im Rahmenwerk für DsNs verwendet), der Mapping-Dienst, der Speicher-Dienst und das DsN. Der Ablauf wird im Folgenden beschrieben.

(0) Zunächst extrahiert das Endsystem aus der URI des angeforderten Dienstes den Dienst-Namen. (1) Danach fragt das Endsystem den Dienst-Namen im Mapping-Dienst ab. Die Abfrage liefert eine Liste von DsNs, die den Dienst anbieten zusammen mit Metadaten wie Kosten, Quality-of-Service oder Service Level Agreements (SLA). (2) Diese Liste wird dem Nutzer präsentiert und der Nutzer muss sich für eines der DsNs entscheiden. (3) Für das ausgewählte DsN wird im Mapping-Dienst die Liste der benötigten Protokolle abgerufen. Für jedes Protokoll wird im Mapping-Dienst die Liste der Bezugsmethoden abgerufen. (4) Danach werden die Protokolle aus dem Speicher-Dienst abgerufen. (5) Nach dem Herunterladen, werden die Protokolle im Rahmenwerk des Endsystems geladen und ausgeführt. (6) Im Mapping-Dienst werden anschließend die möglichen Verbindungsmethoden und Zugangspunkte abgerufen. (7) Mit Hilfe dieser Informationen baut das Endsystem eine Virtuelle Verbindung zu dem DsN auf. Abschließend wird das Zugriffsverfahren beendet und die eigentliche Kommunikation mit dem angeforderten Dienst findet im Rahmenwerk über die neue Virtuelle Verbindung und die neuen Protokolle statt.

In den folgenden Abschnitten werden der Ablauf und die Komponenten des Zugriffsverfahrens näher erläutert:

(1) Das Finden und die Wahl von DsNs wird in Abschnitt 3.4 vorgestellt.

(2) Das Beziehen der benötigten Dienst-spezifischen Protokolle wird in Abschnitt 3.5 behandelt.

(3) Das Verbinden zu DsNs wird in Abschnitt 3.6 präsentiert.

3.4 Finden und Wahl Dienst-spezifischer Netze

Greift ein Nutzer auf einen Dienst zu, muss das Endsystem ein DsN für diesen Dienst finden. Dies bedeutet, das Endsystem muss die DsNs finden, die den Dienst anbieten, und eines dieser DsNs für den Zugriff auf den Dienst auswählen. Das Zugriffsverfahren EUNA erlaubt das Finden und die Wahl von DsNs für einen Dienst. Es gibt Dienste-Anbietern durch ein Namensschema und durch die Verwendung des Mapping-Dienstes die Flexibilität, beliebige Dienste in DsNs anzubieten. Außerdem gibt es dem Nutzer Kontrolle darüber, zu welchen DsNs Verbindungen aufgebaut werden.

Dienste-Anbieter

Das Zugriffsverfahren nutzt, wie weiter oben beschrieben, URI-basierte Namen, um auf Dienste zuzugreifen. Ein Dienste-Anbieter, der einen Dienst in einem DsN ausbringt, spezifiziert ein Namensschema für den Dienst. Dabei legt der Dienste-Anbieter den Dienst-Namen und den Dienst-spezifischen Teil der URI fest. Im Mapping-Dienst wird eine Abbildung von dem Dienst-Namen auf eine Liste von DsNs gespeichert. Die Abbildung wird von Dienste-Anbietern für den entsprechenden Dienst angelegt und von Endsystemen abgerufen. Das Namensschema und der Mapping-Dienst geben Dienste-Anbietern die Möglichkeit, beliebige Dienste zu definieren und diese einfach anzubieten bzw. für Endsysteme zugänglich zu machen.

Bieten mehrere Dienste-Anbieter den selben Dienst an, müssen diese sich auf ein gemeinsames Namensschema einigen. Beim Anlegen der Abbildung vom Dienst-Namen auf die Liste der DsNs müssen die Dienste-Anbieter darauf achten, die Konsistenz des entsprechenden Eintrages im Mapping-Dienst zu erhalten, damit Endsysteme eine vollständige und fehlerfreie Liste abrufen können. Außerdem müssen sie sicherstellen, dass in ihren DsNs die gleichen Dienste von Endsystemen genutzt werden können. Diese Probleme waren allerdings nicht im Fokus dieser Arbeit.

Endsystem

Greift ein Nutzer auf einem Endsystem auf einen Dienst zu, erhält die Komponente *EUNA Control* den Namen des Dienstes über das Rahmenwerk. *EUNA Control* nutzt den *Query Manager*, um den Dienst-Namen im Mapping-Dienst abzufragen. Hierdurch erhält *EUNA Control* eine Liste der DsNs, die den Dienst anbieten, zusammen mit Meta-Daten. Um eines der DsNs auszuwählen, präsentiert *EUNA Control* dem Nutzer die Liste der DsNs über einen separaten Kanal. Der Nutzer wählt ein Netz basierend auf den Meta-Daten aus. Hierdurch besitzt der Nutzer die Kontrolle darüber, zu welchen DsNs Verbindungen aufgebaut werden.

In Abbildung 3.6 wird das Finden und die Wahl eines DsN für einen Dienst auf einem Endsystem anhand eines Sequenzdiagrammes dargestellt. Es werden die beteiligten Komponenten *EUNA Control*, *Query Manager*, *Abfrageprotokoll* und *Mapping-Dienst* dargestellt. Welche Operationen die Komponenten über den Verlauf der Zeit durchführen und welche Informationen welche Komponenten miteinander austauschen, wird mit Hilfe von Pfeilen dargestellt. Um die Darstellung zu vereinfachen, benötigen Operationen und der Informationsaustausch zwischen Komponenten in der Abbildung keine Zeit.

Im Folgenden wird näher auf den in der Abbildung dargestellten Ablauf des Findens und der Wahl eines DsN für einen Dienst eingegangen. Hierfür wird der Ablauf in die drei Schritte Vorarbeiten, Abfragen der DsNs und Wahl eines DsN gegliedert.

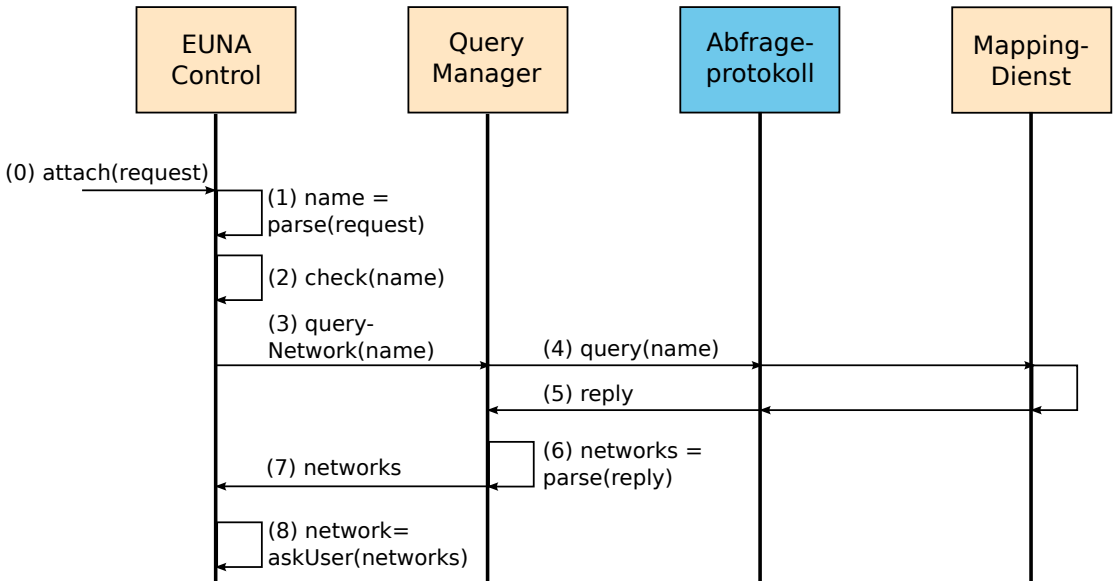

Abbildung 3.6: *Finden und Wahl Dienst-spezifischer Netze für einen Dienst*

3.4.1 Vorarbeiten

Erhält *EUNA Control* vom Rahmenwerk die Aufforderung, auf ein DsN für einen Dienst zuzugreifen (*attach(request)*), führt *EUNA Control* zunächst folgende Vorarbeiten durch. *EUNA Control* ermittelt, (i) auf welchen Dienst der Nutzer zugreifen will und (ii) ob bereits eine Verbindung zu einem DsN für diesen Dienst besteht.

(1) Im ersten Schritt ermittelt *EUNA Control*, auf welchen Dienst der Nutzer zugreifen will. Hierzu wird der angeforderte Name (*request*), der in der über die Schnittstelle erhaltenen Aufforderung enthalten ist, durch *EUNA Control* verarbeitet (*name = parse(request)*). Dabei wird der Dienst-Name (*name*) extrahiert. Der Dienst-Name ist, wie im Namensschema beschrieben, durch einen Doppelpunkt vom Rest des angeforderten Namens *request* getrennt. Falls *request* beispielsweise *web://streamprovider/test* ist, ist *web* der Dienst-Name und *//streamprovider/test* der Dienst-spezifische Teil des Namens.

(2) Nach dem Extrahieren des Dienst-Namens ermittelt *EUNA Control* im zweiten Schritt, ob auf ein DsN zugegriffen werden muss, um auf den angeforderten Dienst zugreifen zu können, oder ob eine bestehende Verbindung zu einem DsN für diesen Dienst verwendet werden kann. Hierfür wird der aus der Aufforderung extrahierte Dienst-Name mit den Dienst-Namen verglichen, die von den DsNs unterstützt werden, zu denen das Endsystem zum aktuellen Zeitpunkt verbunden ist (*check(name)*). Diese Informationen sind in der Liste der DsNs in *EUNA Control* gespeichert. Unterstützt keines der DsNs den Dienst-Namen, muss sich *EUNA Control* mit einem neuem DsN verbinden und fährt daher mit dem Zugriffsverfahren fort. Unterstützt mindestens eines der DsNs den Dienst-Namen, muss *EUNA Control* keine neue Verbindung aufbauen. In

diesem Fall beendet *EUNA Control* das Zugriffsverfahren. Hat eine Anwendung besondere Anforderungen an einen Dienst oder an den Zugriff auf einen Dienst, kann es auch in diesem Fall sinnvoll sein, eine Verbindung mit einem anderen DsN aufzubauen, das geeigneter für die Anforderungen der Anwendung ist. Dieser Fall wurde aber im Rahmen dieser Arbeit nicht näher betrachtet.

3.4.2 Abfragen der Dienst-spezifischen Netze

Hat *EUNA Control* ermittelt, dass auf ein DsN zugegriffen werden muss, um den vom Nutzer angeforderten Dienst zu nutzen, muss *EUNA Control* DsNs finden, die den Dienst anbieten. Daher ruft *EUNA Control* in diesem Schritt die Liste der DsNs, die den Dienst anbieten, aus dem Mapping-Dienst ab. Hierfür werden der *Query Manager*, das *Abfrageprotokoll* und der Mapping-Dienst verwendet (siehe Abbildung 3.6).

(3) Um die Liste der DsNs für einen Dienst abzurufen, nutzt *EUNA Control* den *Query Manager*. *EUNA Control* erteilt dem *Query Manager* über *queryNetworks(name)* die Aufgabe, den Dienst-Namen *name* in eine Liste von DsNs aufzulösen. Der *Query Manager* ist dafür verantwortlich, den Dienst-Namen mit Hilfe des *Abfrageprotokolls* aus dem Mapping-Dienst abzufragen. (4) Der *Query Manager* führt die Abfrage des Dienst-Namens *name* im Mapping-Dienst durch (*query(name)*). (5) Das Abfrageprotokoll liefert die Antwort des Mapping-Dienstes (*reply*) an den *Query Manager* zurück. (6) Der *Query Manager* verarbeitet die Antwort des Abfrageprotokolls weiter (*parse(reply)*). Dabei extrahiert der *Query Manager* die Liste der DsNs (*networks*) aus der Antwort. (7) Diese Liste gibt der *Query Manager* schließlich zurück an *EUNA Control*.

In Tabelle 3.1 wird der vom Endsystem aus dem Mapping-Dienst abgerufene Eintrag für die Liste der DsNs dargestellt. In der linken Spalte werden die Felder des Eintrages, in der rechten Spalte kurze Erklärungen der Felder dargestellt. Der Eintrag im Mapping-Dienst besteht aus den Feldern *Version*, *Name*, *ID*, *Dienst-spezifisches Netz*, *Primitive* und *Anforderungen*. Das Feld *Dienst-spezifisches Netz* besteht aus den zusätzlichen Feldern *ID*, *SLA* und *Kosten*. Der Eintrag enthält mindestens ein Feld *Dienst-spezifisches Netz*, kann aber auch mehrere enthalten, sofern mehrere DsNs den entsprechenden Dienst anbieten.

Das Feld *Version* spezifiziert die Version des Eintrages im Mapping-Dienst und vereinfacht Veränderungen des Eintrages zu erkennen. Das Feld *Name* gibt den Namen des Dienstes an. Das Feld *ID* gibt einen im Mapping-System eindeutigen Identifikator des Dienstes an. Der Eintrag im Mapping-Dienst besteht aus mindestens einem Feld *Dienst-spezifisches Netz*. Diese Felder geben eine Liste von DsNs an, die den Dienst unterstützen. Das Feld *Dienst-spezifisches Netz* gibt den Namen des DsN an und besteht aus weiteren Feldern. Das Feld *ID* gibt eine im Mapping-Dienst eindeutige Identifikationsnummer des DsN an. Im Feld *SLA* werden die Nutzungsbedingungen und Eigenschaften des DsN in Menschen-lesbarer Form abgespeichert. Im Feld *Kosten* werden die dem

Tabelle 3.1: *Eintrag für Dienst-spezifische Netze für einen Dienst im Mapping-Dienst*

Feld	**Beschreibung**
Version	Version des Eintrages
Name	Name des Dienstes
ID	Identifikator des Dienstes
Dienst-spezifisches Netz	Dienst-spezifisches Netz für den Dienst
→ ID	Identifikator des Dienst-spezifischen Netzes
→ SLA	Nutzungsbedingungen des Dienst-spezifischen Netzes
→ Kosten	Kosten für Nutzung des Dienst-spezifischen Netzes
...	Optional: Weitere Dienst-spezifische Netze
Primitive	Vom Dienst unterstützte Primitive des Rahmenwerks für Dienst-spezifische Netze
Anforderungen	Vom Dienst unterstützte Anwendungsanforderungen

Benutzer durch die Nutzung des DsN entstehenden Kosten gespeichert. Die weiteren Felder des Eintrages *Primitive* und *Anforderungen* geben die vom Dienst unterstützten Primitive der Anwendungsschnittstelle des Rahmenwerkes (z.B. *get* und *connect*) und Anwendungsanforderungen an.

3.4.3 Wahl eines Dienst-spezifischen Netzes

Hat *EUNA Control* die Liste der DsNs für den Dienst-Namen erhalten, muss ein DsN ausgewählt werden, zu dem eine Verbindung aufgebaut werden soll. In diesem Schritt wählt daher *EUNA Control* eines der DsNs aus und bezieht dabei den Nutzer in die Entscheidung mit ein (siehe Schritt (8) in Abbildung 3.6).

(8) *EUNA Control* verarbeitet die Liste der DsNs, die vom *Query Manager* zurückgegeben worden ist, weiter. Dabei extrahiert *EUNA Control* die einzelnen DsNs sowie deren Nutzungsbedingungen und Kosten. *EUNA Control* präsentiert die zur Verfügung stehenden DsNs mit ihren Namen, Nutzungsbedingungen und Kosten dem Nutzer (*askUser(networks)*). Dies verläuft über einen separaten Kommunikationskanal, wie z.B. über ein grafisches Pop-up Fenster mit Hilfe eines Betriebssystem-Mechanismus. Basierend auf den präsentierten Informationen muss der Nutzer eines der DsNs auswählen. Die Wahl des Nutzers (*network*) wird zurück an *EUNA Control* übertragen. Damit ist *EUNA Control* bekannt, zu welchem DsN eine Verbindung aufgebaut werden muss.

Alternativ kann die Entscheidung, zu welchem Netz eine Verbindung aufgebaut werden soll, auch automatisiert durch einen separaten Prozess erfolgen, den ein Nutzer durch

Richtlinien parametrisiert. Diese Möglichkeit wurde allerdings im Rahmen dieser Arbeit nicht weiter verfolgt.

3.5 Beziehen der benötigten Dienst-spezifischen Protokolle

Hat der Nutzer ein DsN für einen Dienst ausgewählt, müssen die für das DsN benötigten Dienst-spezifischen Protokolle ermittelt und bezogen werden. Das Zugriffsverfahren EUNA erlaubt es Endsystemen, die Dienst-spezifischen Protokolle transparent für den Nutzer zu beziehen. EUNA gibt Dienste-Anbietern die Kontrolle darüber, welche Dienst-spezifischen Protokolle in ihren DsNs benutzt werden, und darüber, wie die Protokolle bezogen werden können. Außerdem bietet das Zugriffsverfahren Endsystemen die Flexibilität, verfügbare Bezugsmethoden für das Beziehen Dienst-spezifischer Protokolle auszuwählen und die Bezugsmethoden auszutauschen.

Dienste-Anbieter

Das Zugriffsverfahren nutzt für das Beziehen der Dienst-spezifischen Protokolle den Mapping-Dienst und den Speicher-Dienst. Der Mapping-Dienst enthält für ein DsN eine Abbildung auf eine Liste der Dienst-spezifischen Protokolle. Außerdem enthält der Mapping Dienst für jedes Dienst-spezifische Protokoll eine Abbildung auf eine Liste von Bezugsmethoden. Die Bezugsmethoden enthalten Informationen darüber, wie das Dienst-spezifische Protokoll aus dem Speicher-Dienst bezogen werden kann. Die Einträge im Mapping-Dienst werden durch den Dienste-Anbieter des DsN angelegt und durch Endsysteme abgerufen. Die Einträge geben dem Dienste-Anbieter die Kontrolle über die im DsN verwendeten Protokolle und darüber, wie die Dienst-spezifischen Protokolle bezogen werden. Im Speicher-Dienst werden die Dienst-spezifischen Protokolle für das DsN vom Dienste-Anbieter abgespeichert. Endsysteme greifen über die vom Dienste-Anbieter festgelegten Bezugsmethoden auf den Speicher-Dienst zu, um die Dienst-spezifischen Protokolle zu beziehen.

Endsystem

Greift ein Endsystem auf ein DsN zu, ruft es die Einträge aus dem Mapping-Dienst ab. Dadurch erhält es die Liste der Protokolle und deren Bezugsmethoden. Das Endsystem wählt eine geeignete Bezugsmethode für das Beziehen jedes Dienst-spezifischen Protokolls. Das Endsystem kann Bezugsmethoden wählen, die auf dem Endsystem verfügbar sind und mit denen ein Datenaustausch über vorhandene DsNs oder das Basisnetz möglich ist. Außerdem ist es möglich, weitere Bezugsmethoden auf dem Endsystem hinzuzufügen oder

die auf dem Endsystem verfügbaren Bezugsmethoden auszutauschen. Hat das Endsystem Bezugsmethoden gewählt, bezieht es die Protokolle und lädt sie im Rahmenwerk.

In Abbildung 3.7 wird das Beziehen der Dienst-spezifischen Protokolle für ein DsN auf einem Endsystem anhand eines Sequenzdiagrammes dargestellt. Es werden die beteiligten Komponenten *EUNA Control*, *Query Manager*, *Abfrageprotokoll*, *Mapping-Dienst*, *Retrieval Manager*, *Bezugsmethode* und *Speicher-Dienst* dargestellt. Welche Operationen die Komponenten über den Verlauf der Zeit durchführen und welche Informationen welche Komponenten miteinander austauschen, wird mit Hilfe von Pfeilen dargestellt. Um die Darstellung zu vereinfachen, benötigen Operationen und der Informationsaustausch zwischen Komponenten in der Abbildung keine Zeit.

Im Folgenden wird näher auf den in der Abbildung dargestellten Ablauf des Beziehens von Dienst-spezifischen Protokollen für ein DsN eingegangen. Hierfür wird der Ablauf in die vier Schritte Abfragen der benötigten Dienst-spezifischen Protokolle, Abfragen der Bezugsmethoden, Beziehen der Dienst-spezifischen Protokolle und Laden der Dienst-spezifischen Protokolle gegliedert.

3.5.1 Abfragen der benötigten Dienst-spezifischen Protokolle

Nachdem ermittelt worden ist, auf welches DsN das Endsystem zugreifen soll, ermittelt *EUNA Control*, welche Dienst-spezifischen Protokolle in dem DsN verwendet werden. *EUNA Control* ruft in diesem Schritt die Liste der Dienst-spezifischen Protokolle, die für das DsN benötigt werden, aus dem Mapping-Dienst ab. Hierfür werden der *Query Manager*, das *Abfrageprotokoll* und der Mapping-Dienst verwendet.

(1) Um die Liste der Dienst-spezifischen Protokolle für ein DsN abzurufen, nutzt *EUNA Control* den *Query Manager*. *EUNA Control* erteilt dem *Query Manager* über *queryProtocols(network)* die Aufgabe, den Namen des DsN (*network*) in eine Liste von Dienst-spezifischen Protokollen aufzulösen. Der *Query Manager* ist dafür verantwortlich, den Namen des DsN mit Hilfe des *Abfrageprotokolls* aus dem Mapping-Dienst abzufragen. (2) Der *Query Manager* führt die Abfrage des Namens des DsN im Mapping-Dienst durch (*query(network)*). (3) Das Abfrageprotokoll liefert die Antwort des Mapping-Dienstes zurück (*reply*). (4) Der *Query Manager* verarbeitet die Antwort des Abfrageprotokolls weiter (*parse(reply)*). Dabei extrahiert er die Liste der Dienst-spezifischen Protokolle (*protocols*) aus der Antwort. (5) Diese Liste gibt der *Query Manager* zurück an *EUNA Control*.

In Tabelle 3.2 wird der vom Endsystem aus dem Mapping-Dienst abgerufene Eintrag für die Liste der Dienst-spezifischen Protokolle dargestellt. In der linken Spalte werden die Felder des Eintrages, in der rechten Spalte kurze Erklärungen der Felder dargestellt. Der Eintrag im Mapping-Dienst besteht aus den Feldern *Version*, *Name*, *ID* und *Protokoll*. Das Feld *Protokoll* besteht aus den zusätzlichen Feldern *ID* und *Optional*. Der Eintrag

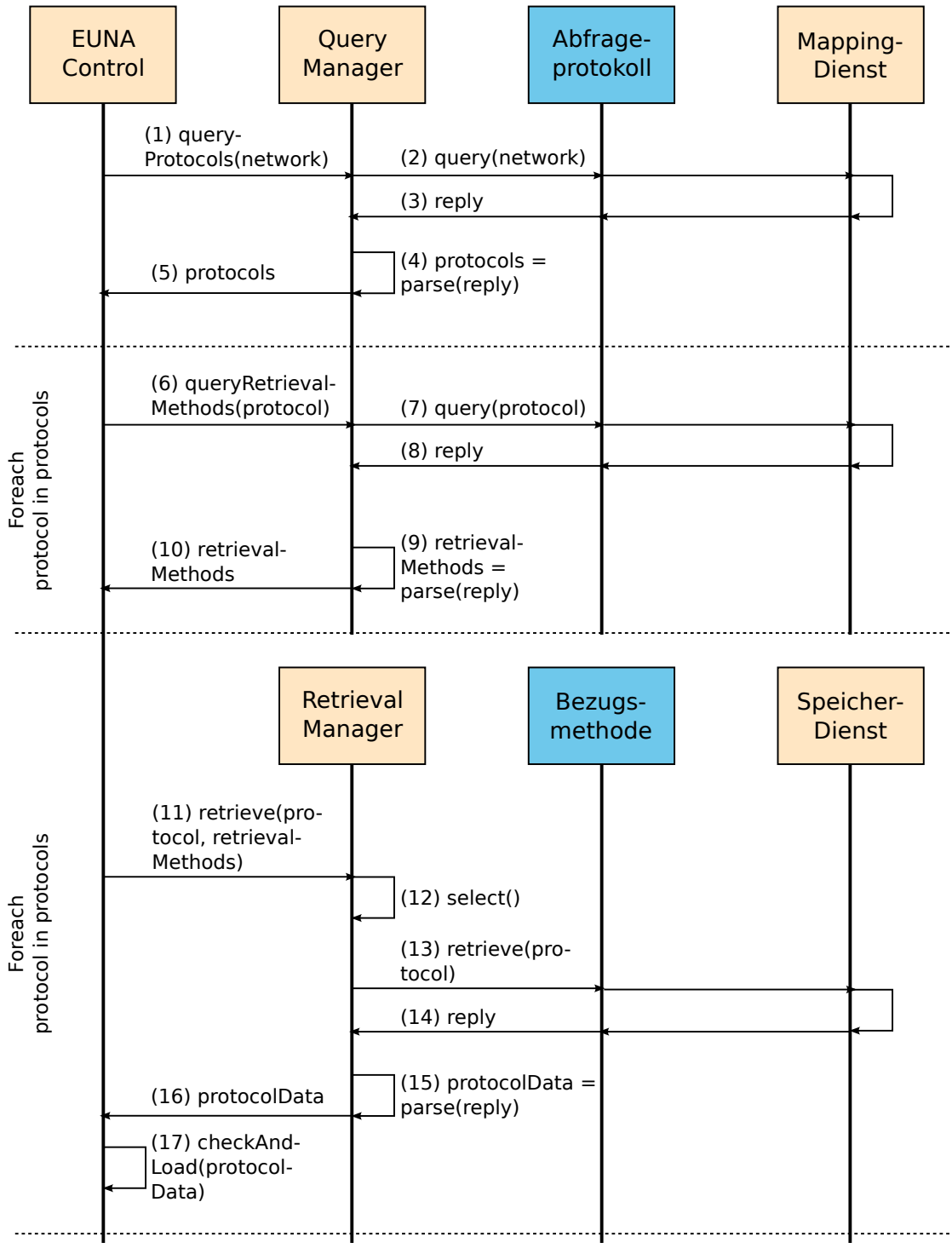

Abbildung 3.7: *Beziehen der Dienst-spezifischen Protokolle für ein Dienst-spezifisches Netz*

enthält mindestens ein Feld *Protokoll*, kann aber auch mehrere enthalten, sofern mehrere Dienst-spezifische Protokolle in dem entsprechenden DsN genutzt werden.

Tabelle 3.2: *Eintrag für Dienst-spezifische Protokolle eines Dienst-spezifischen Netzes im Mapping-Dienst*

Feld	**Beschreibung**
Version	Version des Eintrages
Name	Name des Dienst-spezifischen Netzes
ID	Identifikator des Dienst-spezifischen Netzes
Protokoll	Dienst-spezifisches Protokoll
→ ID	Identifikator des Dienst-spezifischen Protokolls
→ Optional	Angabe, ob Dienst-spezifisches Protokoll optional ist
...	Optional: Weitere Dienst-spezifische Protokolle

Das Feld *Version* gibt die Version des Eintrages an und erlaubt eine Versionierung. Die Felder *Name* und *ID* spiegeln den Namen und die Identifikationsnummer des DsN wider. Der Eintrag im Mapping-Dienst besteht aus mindestens einem Feld *Protokoll*. Diese Einträge geben eine Liste von Dienst-spezifischen Protokollen an, die in dem DsN benutzt werden. Das Feld *Protokoll* gibt den Namen des Dienst-spezifischen Protokolls an und besteht aus weiteren Feldern. Im Feld *ID* wird die im Mapping-Dienst eindeutige Identifikationsnummer des Dienst-spezifischen Protokolls angegeben. Das Feld *Optional* gibt an, ob das Protokoll zwingend notwendig ist innerhalb des Netzes. Ein Dienst-spezifisches Protokoll, das nicht notwendig ist, muss nicht vom Endsystem geladen werden. Ein notwendiges Protokoll muss vom Endsystem geladen werden. Die Reihenfolge, in der die Dienst-spezifischen Protokolle in dem Eintrag aufgeführt werden, entspricht der Reihenfolge, in der die Dienst-spezifischen Protokolle von dem Endsystem im Rahmenwerk instantiiert werden.

3.5.2 Abfragen der Bezugsmethoden

Nachdem *EUNA Control* die Liste der Dienst-spezifischen Protokolle ermittelt hat, müssen diese Dienst-spezifischen Protokolle von dem Endsystem bezogen werden. Dafür benötigt *EUNA Control* Informationen darüber, wie diese bezogen werden können. Daher ruft *EUNA Control* in diesem Schritt für jedes Dienst-spezifische Protokoll die Liste der Bezugsmethoden aus dem Mapping-Dienst ab. Hierfür werden der *Query Manager*, das *Abfrageprotokoll* und der Mapping-Dienst verwendet (siehe Abbildung 3.7).

(6) Um die Liste der Bezugsmethoden für ein Dienst-spezifisches Protokoll abzurufen, nutzt *EUNA Control* den *Query Manager*. *EUNA Control* erteilt dem *Query Manager*

über *queryRetrievalMethods(protocol)* die Aufgabe, den Namen des Dienst-spezifischen Protokolls (*protocol*) in eine Liste von Bezugsmethoden aufzulösen. Der *Query Manager* ist dafür verantwortlich, den Namen des Dienst-spezifischen Protokolls mit Hilfe des *Abfrageprotokolls* aus dem Mapping-Dienst abzufragen. (7) Der *Query Manager* führt die Abfrage des Namens des Dienst-spezifischen Protokolls im Mapping-Dienst durch (*query(protocol)*). (8) Das Abfrageprotokoll liefert die Antwort des Mapping-Dienstes (*reply*) zurück. (9) Der *Query Manager* verarbeitet die Antwort des Abfrageprotokolls weiter (*parse(reply)*). Dabei extrahiert er die Liste der Bezugsmethoden (*retrievalMethods*) aus der Antwort. (10) Diese Liste gibt der *Query Manager* zurück an *EUNA Control*.

In Tabelle 3.3 wird der vom Endsystem aus dem Mapping-Dienst abgerufene Eintrag für die Liste der Bezugsmethoden eines Dienst-spezifischen Protokolls dargestellt. In der linken Spalte werden die Felder des Eintrages, in der rechten Spalte kurze Erklärungen der Felder dargestellt. Der Eintrag im Mapping-Dienst besteht aus den Feldern *Version*, *Name*, *ID*, *Bezugsmethode* und *Selektionsmethode*. Das Feld *Bezugsmethode* besteht aus den zusätzlichen Feldern *Bezugspunkt* und *Selektionsmethode*. Das Feld enthält mindestens einen *Bezugspunkt*. Der Eintrag enthält mindestens ein Feld *Bezugsmethode*, kann aber auch mehrere enthalten, sofern mehrere Bezugsmethoden für das entsprechende Dienst-spezifische Protokoll genutzt werden können.

Tabelle 3.3: *Eintrag für Bezugsmethoden eines Dienst-spezifischen Protokolls im Mapping-Dienst*

Feld	**Beschreibung**
Version	Version des Eintrages und Protokolls
Name	Name des Dienst-spezifischen Protokolls
ID	Identifikator des Dienst-spezifischen Protokolls
Bezugsmethode	Bezugsmethode für das Dienst-spezifischen Protokoll
→ Bezugspunkt	Bezugspunkt für Dienst-spezifisches Protokoll
→ ...	Optional: Weitere Bezugspunkte
→ Selektionsmethode	Entscheidungskriterium für Bezugspunkt
...	Optional: Weitere Bezugsmethoden
Selektionsmethode	Entscheidungskriterium für Bezugsmethode

Das Feld *Version* spezifiziert die Version des Eintrages im Mapping-Dienst und des Protokolls. Hiermit kann ein Endsystem beispielsweise ermitteln, ob ein bereits geladenes Protokoll in einer neueren Version existiert. Die Felder *Name* und *ID* spiegeln den Namen und die Identifikationsnummer des Dienst-spezifischen Protokolls wider. Der Eintrag im Mapping-Dienst besteht aus mindestens einem Feld *Bezugsmethode*. Diese Felder

realisieren die Liste der Bezugsmethoden. Jedes dieser Felder spezifiziert eine Bezugsmethode und besteht aus den weiteren Feldern *Bezugspunkt* und *Selektionsmethode*. *Bezugspunkt* spezifiziert eine Möglichkeit, das Dienst-spezifische Protokoll mit Hilfe der Bezugsmethode zu beziehen. Ist die Bezugsmethode beispielsweise das Protokoll HTTP, kann ein Bezugspunkt eine URL sein. Das Feld *Bezugsmethode* besteht aus mindestens einem Feld *Bezugspunkt*. Das Feld *Selektionsmethode* gibt an, wie einer der Bezugspunkte vom Endsystem ausgewählt werden soll. Da der Eintrag im Mapping-Dienst mehrere *Bezugsmethode* Felder enthalten kann, enthält der Eintrag auch ein Feld *Selektionsmethode*. Dieses Feld gibt an, wie das Endsystem eine der Bezugsmethoden auswählen soll. Beispiele von Selektionsmethoden für Bezugspunkte oder Bezugsmethoden sind eine Auswahl nach dem Zufallsprinzip oder entsprechend der Reihenfolge im Eintrag.

3.5.3 Beziehen der Dienst-spezifischen Protokolle

Nachdem *EUNA Control* für jedes Dienst-spezifische Protokoll eine Liste von Bezugsmethoden erhalten hat, bezieht *EUNA Control* die Dienst-spezifischen Protokolle für das DsN. Hierfür verwendet *EUNA Control* den *Retrieval Manager*. Der *Retrieval Manager* wählt eine Bezugsmethode für jedes Dienst-spezifische Protokoll und bezieht es jeweils über die ausgewählte Bezugsmethode (siehe Abbildung 3.7).

(11) *EUNA Control* erteilt dem *Retrieval Manager* über *retrieve(protocol, retrievalMethods)* die Aufgabe, das Dienst-spezifische Protokoll *protocol* mit den Bezugsmethoden in *retrievalMethods* zu beziehen. Der *Retrieval Manager* ist dafür verantwortlich, das Dienst-spezifische Protokoll mit Hilfe von *Bezugsmethoden* aus dem Speicher-Dienst abzurufen. (12) Basierend auf den in *retrievalMethods* enthaltenen Bezugsmethoden und der Liste der auf dem Endsystem verfügbaren Bezugsmethoden wählt der *Retrieval Manager* eine Bezugsmethode aus (*select()*). Hierbei werden beispielsweise auf dem Endsystem nicht verfügbare Bezugsmethoden gefiltert. Außerdem werden die in den *retrievalMethods* enthaltenen Selektionsmethoden beachtet. Soll beispielsweise eine der Bezugsmethoden zufallsgesteuert gewählt werden, wird aus den auf dem Endsystem verfügbaren Bezugsmethoden eine zufällige ausgewählt. Außerdem wählt der *Retrieval Manager* für die gewählte Bezugsmethode mit Hilfe ihrer Selektionsmethode einen der Bezugspunkte aus. Legt diese Selektionsmethode beispielsweise fest, dass ein Bezugspunkt zufallsgesteuert ausgewählt werden soll, wählt der *Retrieval Manager* einen zufälligen Bezugspunkt aus der Liste der Bezugspunkte aus. (13) Der *Retrieval Manager* führt anschließend das Beziehen des Dienst-spezifischen Protokolls aus dem Speicher-Dienst mit der gewählten Bezugsmethode und dem gewählten Bezugspunkt durch (*retrieve(protocol)*). (14) Die Bezugsmethode liefert die Antwort des Speicher-Dienstes (*reply*) an den *Retrieval Manager* zurück. (15) Der *Retrieval Manager* verarbeitet die Antwort der Bezugsmethode weiter (*parse(reply)*). Dabei extrahiert er das Dienst-spezifische Protokoll bzw. dessen Daten (*protocolData*) aus der Antwort. (16) Diese Daten gibt der *Retrieval Manager*

zurück an *EUNA Control*. (17) *EUNA Control* überprüft schließlich das Dienst-spezifische Protokoll und lädt es im Rahmenwerk (*checkAndLoad(protocolData)*). Dieser Schritt wird genauer im folgenden Abschnitt 3.5.4 behandelt.

Auf diese Weise wird für jedes Dienst-spezifische Protokoll eine Bezugsmethode sowie ein Bezugspunkt ausgewählt und damit das Dienst-spezifische Protokoll aus dem Speicher-Dienst bezogen.

3.5.4 Überprüfen der Dienst-spezifischen Protokolle

Hat das Endsystem Dienst-spezifische Protokolle für ein DsN bezogen, werden diese auf dem Endsystem geladen und ausgeführt. Hierdurch wird fremder Programm-Code auf dem Endsystem ausgeführt. Um die Stabilität des Endsystems sicherzustellen, muss daher überprüft werden, dass die Dienst-spezifischen Protokolle keine Fehler enthalten. Fehler könnten beim Speichern der Dienst-spezifischen Protokolle im Speicher-Dienst und beim Beziehen der Dienst-spezifischen Protokolle aus dem Speicher-Dienst aufgetreten sein. Außerdem besitzen weder der Dienste-Anbieter noch das Endsystem Kontrolle über den Speicher-Dienst. Daher muss ausgeschlossen werden, dass die Dienst-spezifischen Protokolle manipuliert oder falsche Dienst-spezifische Protokolle für das DsN bezogen worden sind. EUNA bietet eine Überprüfung von Dienst-spezifischen Protokollen, die auf asymmetrischer Kryptographie basiert. Der Dienste-Anbieter signiert die Dienst-spezifischen Protokolle seines DsN mit einem privaten Schlüssel des Netzes. Der öffentliche Schlüssel des Netzes wird im Mapping-Dienst publiziert. Das Endsystem nutzt den öffentlichen Schlüssel und die Signaturen, um sicherzustellen, dass die richtigen Dienst-spezifischen Protokolle bezogen worden sind und diese unverändert sind.

Dienste-Anbieter

Wenn ein Dienste-Anbieter ein DsN für einen Dienst erstellt, legt er fest, welche Dienst-spezifischen Protokolle in dem DsN verwendet werden. Um eine Überprüfung der Protokolle durch das Endsystem zu ermöglichen, generiert der Dienste-Anbieter ein kryptographisches Schlüssel-Paar für das DsN. Der Dienste-Anbieter erstellt ein eigenes Schlüssel-Paar für jedes DsN, das er erstellt. Hierdurch wird verhindert, dass z.B. durch die Offenlegung eines privaten Schlüssels alle DsNs des Dienste-Anbieters betroffen sind. Der Dienste-Anbieter signiert die Dienst-spezifischen Protokolle des DsN mit dem privaten Schlüssel. Um die Signaturen den Endsystemen bereit zu stellen, legt der Dienste-Anbieter die Dienst-spezifischen Protokolle zusammen mit ihren Signaturen im Speicher-Dienst ab. Damit Endsysteme die Dienst-spezifischen Protokolle anhand ihrer Signaturen überprüfen können, benötigen sie den öffentlichen Schlüssel des DsN. Daher publiziert der Dienste-Anbieter den öffentlichen Schlüssel im Mapping-Dienst. Hierfür wird eine Abbildung des DsN auf den öffentlichen Schlüssel im Mapping-Dienst angelegt.

Es wird davon ausgegangen, dass der Mapping-Dienst vertrauenswürdig ist und ein sicheres Speichern und Abrufen von öffentlichen Schlüsseln erlaubt. Um das Austauschen des öffentlichen Schlüssels durch einen potentiellen Angreifer zu verhindern, darf nur der Dienste-Anbieter, der diesen Eintrag im Mapping-Dienst-angelegt hat, diesen auch ändern. Außerdem wird davon ausgegangen, dass die Kommunikation zwischen Dienste-Anbieter sowie Endsystemen und dem Mapping-Dienst durch geeignete Verfahren vor Manipulationen gesichert ist.

Endsystem

Das Endsystem ruft beim Zugriff auf das DsN des Dienste-Anbieters die Dienst-spezifischen Protokolle aus dem Speicher-Dienst ab. Dabei werden die Signaturen der Dienst-spezifischen Protokolle bezogen, die der Dienste-Anbieter im Speicher-Dienst abgelegt hat. Wenn das Endsystem die Dienst-spezifischen Protokolle für das DsN bezogen hat, muss es diese überprüfen, bevor sie geladen werden, um Fehler und Manipulationen auszuschließen. Daher werden im Schritt *checkAndLoad()* in der Abbildung 3.7 die folgenden Schritte durchgeführt. Für die Überprüfung der Dienst-spezifischen Protokolle ruft das Endsystem den öffentlichen Schlüssel aus dem Mapping-Dienst ab. Anschließend überprüft das Endsystem die Dienst-spezifischen Protokolle mit Hilfe der Signaturen und des öffentlichen Schlüssel des DsN. Ist die Überprüfung erfolgreich, werden die Dienst-spezifischen Protokolle geladen.

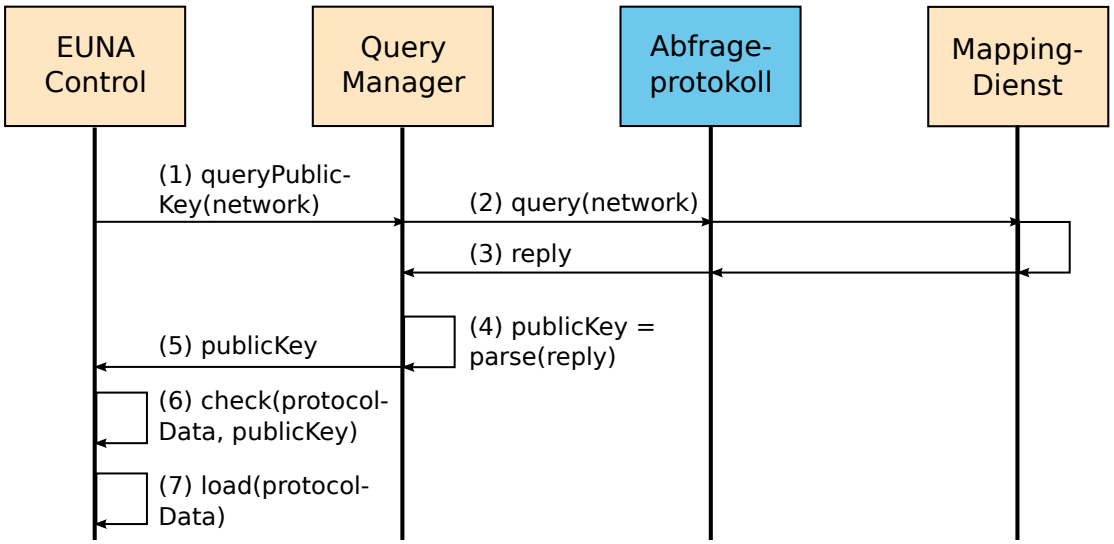

Abbildung 3.8: *Überprüfen und Laden eines Dienst-spezifischen Protokolls*

In Abbildung 3.8 wird das Überprüfen eines Dienst-spezifischen Protokolls auf einem Endsystem anhand eines Sequenzdiagrammes dargestellt. Es werden die beteiligten

Komponenten *EUNA Control*, *Query Manager*, das *Abfrageprotokoll* und der *Mapping-Dienst* dargestellt. Welche Operationen die Komponenten über den Verlauf der Zeit durchführen und welche Informationen welche Komponenten miteinander austauschen, wird mit Hilfe von Pfeilen dargestellt. Um die Darstellung zu vereinfachen, benötigen Operationen und der Informationsaustausch zwischen Komponenten in der Abbildung keine Zeit.

Im Folgenden wird näher auf den in der Abbildung dargestellten Ablauf des Überprüfens Dienst-spezifischer Protokolle eingegangen. Hierfür wird der Ablauf in die zwei Schritte Abrufen des öffentlichen Schlüssels und Überprüfen der Protokolle gegliedert.

3.5.4.1 Abrufen des öffentlichen Schlüssels

Nachdem ein Dienst-spezifisches Protokoll bezogen worden ist, wird es von *EUNA Control* überprüft. Dafür benötigt *EUNA Control* den öffentlichen Schlüssel des DsN. *EUNA Control* ruft diesen aus dem Mapping-Dienst ab. Hierfür werden der *Query Manager*, das *Abfrageprotokoll* und der Mapping-Dienst verwendet. Dieser Schritt muss nur ein mal pro Zugriff auf ein DsN durchgeführt werden. Besitzt das Endsystem den öffentlichen Schlüssel des DsN, können damit alle bezogenen Dienst-spezifischen Protokolle des entsprechenden DsN überprüft werden.

(1) Um den öffentlichen Schlüssel des DsN abzurufen, nutzt *EUNA Control* den *Query Manager*. *EUNA Control* erteilt dem *Query Manager* über *queryPublicKey(network)* die Aufgabe, den Namen des DsN (*network*) in den entsprechenden öffentlichen Schlüssel aufzulösen. Der *Query Manager* ist dafür verantwortlich, den Namen des DsN mit Hilfe des *Abfrageprotokolls* aus dem Mapping-Dienst abzufragen. (2) Der *Query Manager* führt die Abfrage des Namens des DsN im Mapping-Dienst durch (*query(network)*). Es wird davon ausgegangen, dass die Übertragung durch das Abfrageprotokoll mit geeigneten Verfahren vor Manipulationen geschützt ist. (3) Das Abfrageprotokoll gibt die Antwort des Mapping-Dienstes (*reply*) an den *Query Manager* zurück. (4) Der *Query Manager* verarbeitet die Antwort des Abfrageprotokolls weiter (*parse(reply)*). Dabei extrahiert er den öffentlichen Schlüssel (*publicKey*) aus der Antwort. (5) Diesen Schlüssel gibt der *Query Manager* zurück an *EUNA Control*.

In Tabelle 3.4 wird der vom Endsystem aus dem Mapping-Dienst abgerufene Eintrag für den öffentlichen Schlüssel des DsN dargestellt. In der linken Spalte werden die Felder des Eintrages, in der rechten Spalte kurze Erklärungen der Felder dargestellt. Der Eintrag im Mapping-Dienst besteht aus den Feldern *Version*, *Name*, *ID* und *Öffentlicher Schlüssel*.

Das Feld *Version* spezifiziert die Version des Eintrages und des öffentlichen Schlüssels. Die Version erlaubt einem Endsystem beispielsweise, zu erkennen, wenn ein neuerer öffentlicher Schlüssel für das DsN existiert. Die Felder *Name* und *ID* spiegeln den Namen und die Identifikationsnummer des DsN wider. Das Feld *Öffentlicher Schlüssel* gibt den öffentlichen Schlüssel des DsN an.

Tabelle 3.4: *Eintrag für einen öffentlichen Schlüssel eines Dienst-spezifischen Netzes im Mapping-Dienst*

Feld	**Beschreibung**
Version	Version des Eintrages
Name	Name des Dienst-spezifischen Netzes
ID	Identifikator des Dienst-spezifischen Netzes
Öffentlicher Schlüssel	Öffentlicher Schlüssel des Dienst-spezifischen Netzes

3.5.4.2 Überprüfen der Dienst-spezifischen Protokolle

Nachdem *EUNA Control* den öffentlichen Schlüssel des DsN aus dem Mapping-Dienst abgefragt hat, können die Dienst-spezifischen Protokolle überprüft werden (vgl. Abbildung 3.8). Hierfür verwendet *EUNA Control* die zusammen mit den Dienst-spezifischen Protokollen bezogenen Signaturen der Dienst-spezifischen Protokolle und den öffentlichen Schlüssel des DsN. (6) *EUNA Control* überprüft die Signatur jedes einzelnen Dienst-spezifischen Protokolls (*check(protocolData, publicKey)*). (7) Stimmen die Signaturen der Dienst-spezifischen Protokolle, werden diese im Rahmenwerk geladen (*load(protocolData)*).

3.6 Verbinden zu Dienst-spezifischen Netzen

Hat der Nutzer ein DsN für einen Dienst ausgewählt, muss eine Virtuelle Verbindung zu dem DsN aufgebaut werden. Hierfür muss das Endsystem ermitteln, wie eine solche Verbindung zu dem DsN aufgebaut werden kann. Das Zugriffsverfahren EUNA erlaubt Endsystemen, eine Virtuelle Verbindung transparent für den Nutzer aufzubauen. Es gibt Dienste-Anbietern die Kontrolle darüber, welche Verbindungsmethoden und Zugangspunkte zu ihren DsNs benutzt werden. Außerdem bietet es Endsystemen die Flexibilität, verfügbare Verbindungsmethoden für das Aufbauen einer Virtuellen Verbindung auszuwählen und die verfügbaren Verbindungsmethoden auszutauschen.

Dienste-Anbieter

Das Zugriffsverfahren nutzt für das Aufbauen Virtueller Verbindungen zu DsNs den Mapping-Dienst und Zugangspunkte. Der Mapping-Dienst enthält für DsNs eine Abbildung auf eine Liste von Verbindungsmethoden und Zugangspunkten. Diese Abbildung wird vom Dienste-Anbieter des DsN angelegt und von Endsystemen abgefragt. Dadurch besitzt der Dienste-Anbieter die Kontrolle über die Verbindungsmethoden und Zugangspunkte zu seinem DsN. Über die Zugangspunkte können Endsysteme Virtuelle Verbin-

dungen mit Hilfe der entsprechenden Verbindungsmethoden aufbauen. Sie werden vom Dienste-Anbieter festgelegt. Hierdurch besitzt der Dienste-Anbieter Kontrolle darüber, wie Endsysteme sich mit seinem DsN verbinden.

Endsystem

Um auf ein DsN zuzugreifen, ruft das Endsystem die Verbindungsmethoden und Zugangspunkte zum DsN aus dem Mapping-Dienst ab. Das Endsystem wählt eine der Verbindungsmethoden und einen Zugangspunkt aus. Die Liste der Verbindungsmethoden erlaubt dem Endsystem, eine geeignete Verbindungsmethode zu finden. Das Endsystem kann Verbindungsmethoden auswählen, die auf dem Endsystem verfügbar sind und mit denen eine Kommunikation über die vorhandene Netz-Infrastruktur, DsNs oder das Basisnetz möglich ist. Die Liste der Zugangspunkte erlaubt einen geeigneten Zugangspunkt zu finden. Vom Dienste-Anbieter gelieferte Entscheidungskriterien können zudem die Wahl von Verbindungsmethoden oder Zugangspunkten beeinflussen. Außerdem ist es möglich, weitere Verbindungsmethoden auf dem Endsystem hinzuzufügen oder die auf dem Endsystem verfügbaren Verbindungsmethoden auszutauschen. Nach der Wahl baut das Endsystem eine Virtuelle Verbindung zu dem DsN über die gewählte Verbindungsmethode und den gewählten Zugangspunkt auf.

In Abbildung 3.9 wird das Verbinden zu einem DsN auf einem Endsystem anhand eines Sequenzdiagrammes dargestellt. Es werden die beteiligten Komponenten *EUNA Control*, *Query Manager*, *Abfrageprotokoll*, *Mapping-Dienst*, *Attachment Manager*, *Verbindungsmethode* und *Zugangspunkt* dargestellt. Welche Operationen die Komponenten über den Verlauf der Zeit durchführen und welche Informationen welche Komponenten miteinander austauschen, wird mit Hilfe von Pfeilen dargestellt. Um die Darstellung zu vereinfachen, benötigen Operationen und der Informationsaustausch zwischen Komponenten in der Abbildung keine Zeit.

Im Folgenden wird näher auf den in der Abbildung dargestellten Ablauf des Verbindens zu einem DsN eingegangen. Hierfür wird der Ablauf in die drei Schritte Abfragen der Verbindungsmethoden, Aufbauen einer Virtuellen Verbindung und Abschließende Arbeiten gegliedert.

3.6.1 Abfragen der Verbindungsmethoden

Nachdem ein DsN ausgewählt worden ist, zu dem eine Verbindung aufgebaut werden soll, muss *EUNA Control* eine Virtuelle Verbindung zu dem DsN aufbauen. Dafür benötigt *EUNA Control* Informationen darüber, wie eine solche Virtuelle Verbindung aufgebaut werden kann. Daher ruft *EUNA Control* in diesem Schritt für das DsN die Liste der Verbindungsmethoden aus dem Mapping-Dienst ab. Hierfür werden der *Query Manager*, das *Abfrageprotokoll* und der Mapping-Dienst verwendet.

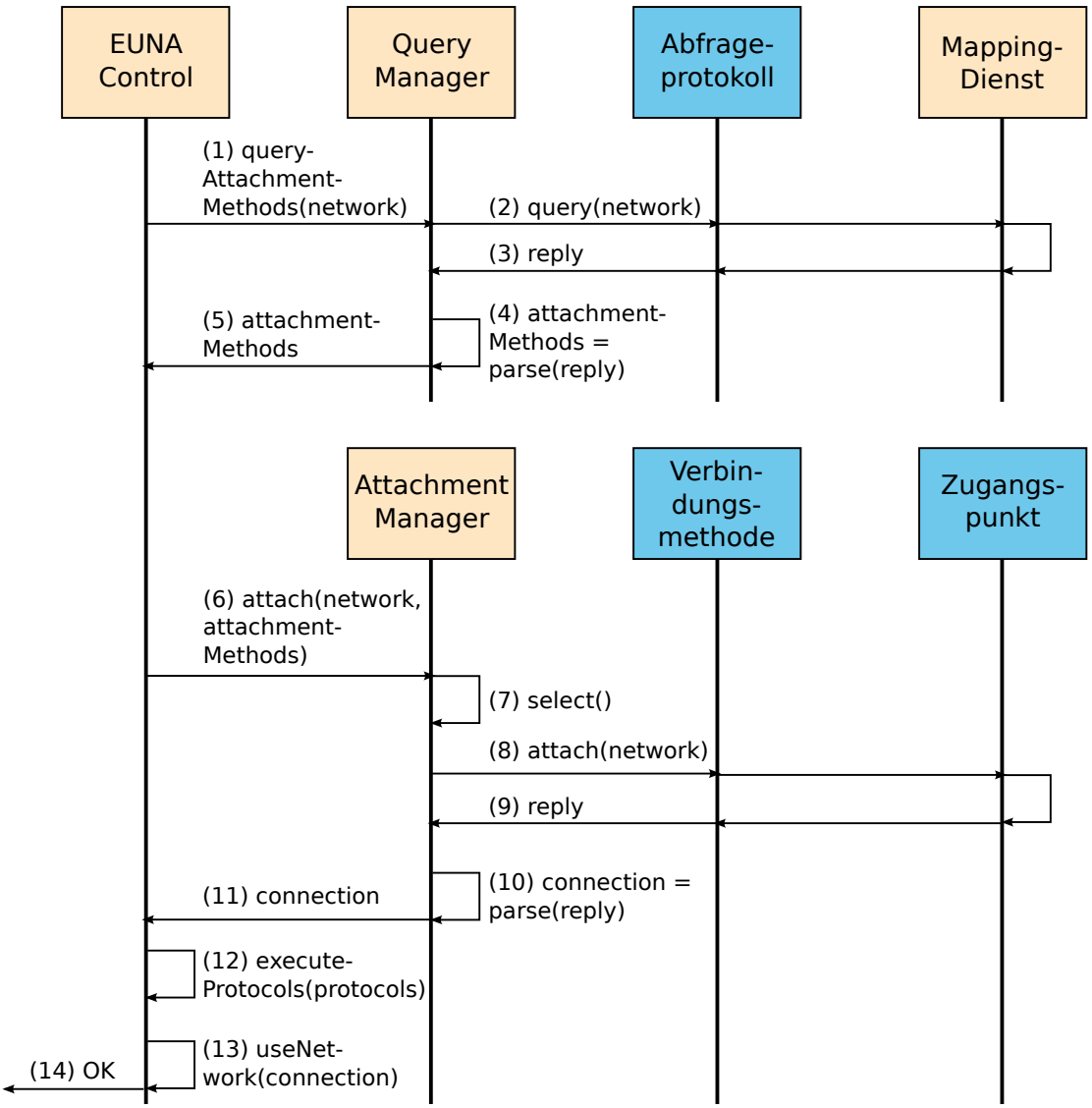

Abbildung 3.9: *Aufbauen einer Virtuellen Verbindung zu einem Dienst-spezifischen Netz*

(1) Um die Liste der Verbindungsmethoden für ein DsN abzurufen, nutzt *EUNA Control* den *Query Manager*. *EUNA Control* erteilt dem *Query Manager* über *queryAttachmentMethods(network)* die Aufgabe, den Namen des DsN (*network*) in eine Liste von Verbindungsmethoden aufzulösen. Der *Query Manager* ist dafür verantwortlich, den Namen des DsN mit Hilfe des *Abfrageprotokolls* aus dem Mapping-Dienst abzufragen. (2) Der *Query Manager* führt die Abfrage des Namens des DsN im Mapping-Dienst durch (*query(network)*). (3) Das Abfrageprotokoll gibt die Antwort des Mapping-Dienstes (*reply*) an den *Query Manager* zurück. (4) Der *Query Manager* verarbeitet die Antwort weiter (*process(reply)*). Dabei extrahiert er die Liste der Verbindungsmethoden (*attachmentMethods*) aus der Antwort. (5) Diese Liste gibt der *Query Manager* zurück an *EUNA Control*.

In Tabelle 3.5 wird der vom Endsystem aus dem Mapping-Dienst abgerufene Eintrag für die Liste der Verbindungsmethoden dargestellt. In der linken Spalte werden die Felder des Eintrages, in der rechten Spalte kurze Erklärungen der Felder dargestellt. Der Eintrag im Mapping-Dienst besteht aus den Feldern *Version*, *Name*, *ID*, *Verbindungsmethode* und *Selektionsmethode*. Das Feld *Verbindungsmethode* besteht aus den zusätzlichen Feldern *Zugangspunkt* und *Selektionsmethode*. Die *Verbindungsmethode* enthält mindestens einen *Zugangspunkt*. Der Eintrag enthält mindestens ein Feld *Verbindungsmethode*, kann aber auch mehrere enthalten, sofern mehrere Verbindungsmethoden zu dem entsprechenden DsN genutzt werden können.

Tabelle 3.5: *Eintrag für Verbindungsmethoden eines Dienst-spezifischen Netzes im Mapping-Dienst*

Feld	**Beschreibung**
Version	Version des Eintrages
Name	Name des Dienst-spezifischen Netzes
ID	Identifikator des Dienst-spezifischen Netzes
Verbindungsmethode	Verbindungsmethode für das Dienst-spezifischen Netz
→ Zugangspunkt	Zugangspunkt zum Dienst-spezifischen Netz
→ ...	Optional: Weitere Zugangspunkte
→ Selektionsmethode	Entscheidungskriterium für Zugangspunkt
...	Optional: Weitere Verbindungsmethoden
Selektionsmethode	Entscheidungskriterium für Verbindungsmethode

Das Feld *Version* spezifiziert die Version des Eintrages. Die Felder *Name* und *ID* spiegeln den Namen und Identifikationsnummer des DsN wider. Dies erlaubt eine Zuordnung der Antwort des Mapping-Systems auf die Anfrage des Endsystems und eine Erkennung von

Fehlern oder Inkonsistenzen im Mapping-Dienst. Der Eintrag im Mapping-Dienst besteht aus mindestens einem Feld *Verbindungsmethode*. Diese Felder realisieren die Liste von Verbindungsmethoden und die entsprechenden Zugangspunkte zu dem DsN. Ein solches Feld spezifiziert eine Verbindungsmethode in das Netz und besteht aus den weiteren Feldern *Zugangspunkt* und *Selektionsmethode*. Das Feld *Zugangspunkt* spezifiziert eine Möglichkeit, zu dem DsN eine Virtuelle Verbindung mit Hilfe der Verbindungsmethode aufzubauen. Wird als Verbindungsmethode beispielsweise ein UDP-Tunnel über IPv4 verwendet, kann ein Zugangspunkt eine IPv4-Adresse und eine UDP-Portnummer sein. Eine *Verbindungsmethode* kann mehrere *Zugangspunkt* Felder enthalten. Das Feld *Selektionsmethode* gibt an, wie einer der Zugangspunkte vom Endsystem ausgewählt werden soll. Der Eintrag im Mapping-Dienst kann mehrere *Verbindungsmethode* Felder enthalten. Das Feld *Selektionsmethode* gibt an, wie das Endsystem eine der Verbindungsmethoden auswählen soll. Beispiele von Selektionsmethoden für Verbindungsmethoden und Zugangspunkte sind eine Auswahl nach dem Zufallsprinzip oder nach der Reihenfolge im Eintrag.

3.6.2 Aufbauen einer Virtuellen Verbindung

Nachdem *EUNA Control* für das DsN eine Liste von Verbindungsmethoden erhalten hat, baut *EUNA Control* die Virtuelle Verbindung zu dem DsN auf. Hierfür verwendet *EUNA Control* den *Attachment Manager*. Der *Attachment Manager* wählt eine Verbindungsmethode für das DsN und baut eine Virtuelle Verbindung mit Hilfe der ausgewählten Verbindungsmethode auf (vgl. Abbildung 3.9).

(6) Um eine Virtuelle Verbindung mit Hilfe der Liste der Verbindungsmethoden aufzubauen, nutzt *EUNA Control* den *Attachment Manager*. *EUNA Control* erteilt dem *Attachment Manager* über *attach(network, attachmentMethods)* die Aufgabe, eine Verbindung zu dem DsN *network* mit den Verbindungsmethoden in *attachmentMethods* aufzubauen. Der *Attachment Manager* ist dafür verantwortlich, die Virtuelle Verbindung zu dem DsN aufzubauen. (7) Basierend auf den in *attachmentMethods* enthaltenen Verbindungsmethoden und der Liste der auf dem Endsystem verfügbaren Verbindungsmethoden wählt der *Attachment Manager* eine Verbindungsmethode (*select()*). Hierbei werden beispielsweise auf dem Endsystem nicht verfügbare Verbindungsmethoden gefiltert. Außerdem werden die in den *attachmentMethods* enthaltenen Selektionsmethoden beachtet. Soll beispielsweise eine der Verbindungsmethoden zufallsgesteuert gewählt werden, wird aus den auf dem Endsystem verfügbaren Verbindungsmethoden eine zufällige ausgewählt. Der *Attachment Manager* wählt außerdem für die gewählte Verbindungsmethode einen Zugangspunkt mit Hilfe ihrer Selektionsmethode aus. Legt beispielsweise diese Selektionsmethode fest, dass ein Zugangspunkt zufallsgesteuert gewählt werden soll, wählt der *Attachment Manager* einen zufälligen Zugangspunkt aus der Liste der Zugangspunkte aus. (8) Der *Attachment Manager* führt anschließend das Aufbauen der Virtuellen

Verbindung zu dem DsN über die gewählte *Verbindungsmethode* und den gewählten *Zugangspunkt* durch (*attach(network)*). (9) Die Verbindungsmethode gibt die Antwort des Zugangspunktes (*reply*) an den *Attachment Manager* zurück. (10) Der *Attachment Manager* verarbeitet die Antwort der Verbindungsmethode weiter (*parse(reply)*). Dabei extrahiert er Informationen, die die Virtuelle Verbindung auf dem Endsystem identifizieren (*connection*), aus der Antwort. (11) Diese Informationen gibt der *Attachment Manager* zurück an *EUNA Control*.

3.6.3 Abschließende Arbeiten

Nachdem *EUNA Control* erfolgreich die Virtuelle Verbindung zu dem DsN aufgebaut hat und in den Schritten zuvor alle benötigten Dienst-spezifischen Protokolle erfolgreich bezogen hat, werden noch die folgenden abschließenden Schritte durchgeführt (vgl. Abbildung 3.9). (12) *EUNA Control* sorgt dafür, dass die Dienst-spezifischen Protokolle im Rahmenwerk für DsNs ausgeführt werden (*executeProtocols(protocols)*). (13) Außerdem teilt *EUNA Control* dem Rahmenwerk die Informationen mit, die das Nutzen der Virtuellen Verbindung im Rahmenwerk ermöglichen (*useNetwork(connection)*). Das Rahmenwerk ist dafür verantwortlich, die geladenen Dienst-spezifischen Protokolle zu der Virtuellen Verbindung zuzuordnen und die geladenen Dienst-spezifischen Protokolle auszuführen. (14) Im letzen Schritt beendet *EUNA Control* das Zugriffsverfahren und schickt eine Bestätigung des erfolgreichen Durchführens (*OK*) an das Rahmenwerk.

3.7 Implementierung

Um die Funktionsfähigkeit von EUNA zu testen und den Aufwand des Zugriffsverfahrens zu evaluieren, wurde ein Prototyp entwickelt. Die Komponenten des Zugriffsverfahrens für DsNs wurden im Prototypen auf folgende Weise realisiert.

3.7.1 Mapping-Dienst

Der Mapping-Dienst wurde mit Hilfe von HiiMap [42] realisiert. HiiMap ist eine hierarchisch organisierte Verteilte Hash-Tabelle (*distributed hash table*, DHT) und basiert auf dem 1-Hop DHT Konzept [81]. In HiiMap wird der Zahlen-Raum auf zwei Hierarchie-Ebenen aufgeteilt. Auf der ersten Hierarchie-Ebene werden Verweise auf die zweite Hierarchie-Ebene gespeichert. Die Zweite Hierarchie-Ebene besteht aus Mapping Regionen, in denen jeweils die Mapping-Einträge verwaltet werden. HiiMap verwendet einen 256 Bit großen Zahlen-Raum und erlaubt eine Abbildung der 256 Bit Schlüssel auf Werte wie z.B. Strings.

Um den Eintrag für einen Dienst im Mapping-Dienst finden zu können, muss der Dienst-Name auf einen 256 Bit Schlüssel abgebildet werden. Hierfür wird das Hash-Verfahren MD5 [93] benutzt und der MD5 Hash des Dienst-Namen gebildet. Da MD5 Hashes eine Länge von 128 Bit besitzen, wird der Hash verdoppelt und konkateniert. Somit ergibt sich der Schlüssel $ID(name) = md5(name)|md5(name)$.

Die Einträge des Zugriffsverfahrens werden als Strings im Mapping-Dienst abgelegt. Der oben beschriebene Schlüssel eines Dienstes wird auf eine Liste von DsNs abgebildet, die jeweils einen eigenen Schlüssel besitzen. Der Schlüssel eines DsN wird auf eine Liste von Protokollen abgebildet, wobei jedes Protokoll wiederum einen eigenen Schlüssel besitzt. Der Schlüssel eines Protokolls wird auf eine Liste von Bezugsmethoden und Bezugspunkten abgebildet. Der Schlüssel eines DsN wird außerdem auf eine Liste von Verbindungsmethoden und Zugangspunkten abgebildet. Die Strings bzw. Einträge werden als Komma-separierte Listen formatiert. Um die Flexibilität zu erhöhen, könnten Datenformate wie XML [16] oder JSON [15] verwendet werden, die zukünftige Erweiterungen oder Veränderungen vereinfachen.

In HiiMap ist eine Public Key Infrastructure (PKI) integriert [41], die auf dem in [102] vorgestellten Verfahren zur Schlüssel-Verteilung basiert. Dies erlaubt das sichere Abrufen von öffentlichen Schlüsseln aus dem Mapping-Dienst.

3.7.2 Speicher-Dienst

Um den Speicher-Dienst zu realisieren, wurden Webserver verwendet. Dabei entspricht jeder Webserver einem Repository, auf dem Protokolle gespeichert werden. Somit ist die Bezugsmethode für die Dienst-spezifischen Protokolle in diesem Prototypen das Protokoll HTTP. Die Dienst-spezifischen Protokolle können von den Webservern über URLs (Bezugspunkte) heruntergeladen werden. HTTP wird im Prototyp über die Protokolle TCP und IPv4 betrieben (siehe Basisnetz).

3.7.3 Basisnetz

Das Basisnetz im Prototypen wurde mit Hilfe des Internets und der G-Lab Experimentierplattform, dem Deutschland-weiten Test-Netz des German-Lab Projektes [32], realisiert. Die Komponenten des Zugriffsverfahrens werden in den G-Lab Standorten ausgebracht und über das Internet für Endsysteme zugänglich gemacht. Da die Protokolle IPv4, UDP und TCP im Internet verfügbar sind, verwendet das Basisnetz diese Protokolle.

3.7.4 Virtuelle Verbindungen

Für die Virtuelle Verbindungen werden UDP-Tunnel über das Basisnetz bzw. das Internet verwendet. Als Verbindungsmethode wird daher ein Challenge/Response Protokoll ba-

sierend auf den Protokollen IPv4 und UDP verwendet. Für die Zugangspunkte zu DsNs wurden in dem Prototypen Challenge/Response Server verwendet. Endsysteme schicken zu diesen Servern über die Verbindungsmethode eine Verbindungsanfrage, die die Server mit einer Nachricht bestätigen.

3.7.5 Rahmenwerk auf Endsystemen

Als Rahmenwerk für die Virtuellen Verbindungen und den Betrieb der Dienst-spezifischen Protokolle wurde das in Abschnitt 5.2 vorgestellte Rahmenwerk NENA verwendet. NENA wurde um die EUNA-Komponenten erweitert. Soll NENA für einen Nutzer eine neue Kommunikationsbeziehung zu einem Dienst aufbauen, wird das Zugriffsverfahren gestartet. EUNA nutzt HiiMap, um die benötigten Informationen aus dem Mapping-Dienst abzurufen. EUNA befragt zudem den Nutzer, ob und zu welchem DsN eine Verbindung aufgebaut werden soll, über die Anwendungsschnittstelle von NENA. Hierfür wurde auch diese erweitert. Um die Dienst-spezifischen Protokolle für ein DsN zu beziehen verwendet EUNA HTTP. Die Protokolle werden von Repositories heruntergeladen und in NENA als dynamische Bibliotheken instantiiert. Die Virtuellen Verbindungen werden über UDP-Tunnel aufgebaut und in NENA als virtuelle Netzwerkschnittstellen (so genannte Network Accesses) verfügbar gemacht. Das Rahmenwerk assoziiert die Dienst-spezifischen Protokolle mit den zugehörigen Virtuellen Verbindungen über einen Multiplexer. Dieser wird zusammen mit den Dienst-spezifischen Protokollen des DsN von Repositories bezogen und in NENA instantiiert.

3.8 Evaluierung

In diesem Abschnitt wird das Zugriffsverfahren EUNA anhand des im vorherigen Abschnitt vorgestellten Prototypen evaluiert. Um die Funktionsfähigkeit des Zugriffsverfahrens zu zeigen, wurde zunächst mit Hilfe des Prototypen ein Demonstrator aufgebaut. Anschließend wurde in weiteren Untersuchungen auf den Zeitaufwand des Zugriffsverfahrens eingegangen.

3.8.1 Demonstrator

Mit Hilfe des Prototypen wurde die Funktionsfähigkeit des Zugriffsverfahren gezeigt. Dazu wurde zunächst ein Demonstrator [123] aufgebaut und auf dem 11ten EuroView Future Internet Workshop präsentiert. Hierfür wurde als Anwendungsfall ein DsN für einen Video-Streaming Dienst entworfen. Der Mapping-Dienst und ein Repository wurden in der G-Lab Experimentierplattform an dem Standort München ausgebracht. Darüber hinaus wurde ein Video-Streaming Server und ein Endsystem lokal am Konferenzort

betrieben. Über eine graphische Benutzeroberfläche legte ein Dienste-Anbieter die im DsN verwendeten Protokolle fest, legte diese auf dem Repository ab und legte die Einträge im Mapping-Dienst an. Der Nutzer auf dem Endsystem verwendete eine Anwendung, die ein Video über den Video-Streaming Dienst empfängt. Auf dem Video-Streaming Server wurde das DsN und der Video-Streaming Dienst bereit gestellt. Beim Zugriff auf den Dienst durch den Nutzer wurde das Zugriffsverfahren gestartet. Dabei wurde dem Nutzer ein Auswahl-Fenster präsentiert, über das er entscheiden konnte, ob eine Verbindung zu dem DsN aufgebaut werden soll. Beim Aufbau der Verbindung wurden automatisch die Dienst-spezifischen Protokolle bezogen und als dynamische Bibliotheken im Rahmenwerk NENA geladen. Anschließend wurden die Virtuelle Verbindung und Dienst-spezifischen Protokolle dazu verwendet, um das Video vom Video-Streaming Server zu empfangen und in der Anwendung des Nutzers abzuspielen.

3.8.2 Zeitaufwand

Um den Zeitaufwand des Zugriffsverfahrens abschätzen zu können, wurden weitere Experimente in der G-Lab Experimentierplattform durchgeführt. Es wurden HiiMap Mapping Regionen bzw. Cluster bestehend aus jeweils vier HiiMap-Knoten in den G-Lab Standorten in den Städten Darmstadt, Karlsruhe und Kaiserslautern platziert. Darüber hinaus wurde ein Webserver als Speicher-Dienst und ein Challenge/Response Server als Zugangspunkt im G-Lab Standort in der Stadt München platziert. Der Zeitaufwand des Zugriffsverfahren wurde gemessen, indem die Dauer der Mapping-Dienst Abfragen aus jeweils zufällig gewählten Clustern, des Herunterladens der Protokolle und des Zugriffs auf den Zugangspunkt von einem Endsystem in Karlsruhe erfasst wurde. Das Endsystem war über eine ADSL Internet Verbindung mit einer maximalen Datenrate von 16 Megabit/s im Downstream und 1 Megabit/s im Upstream an das Internet angeschlossen. Alle Schritte wurden dabei sequentiell vom Endsystem durchgeführt.

In Abbildung 3.10 werden die Ergebnisse der experimentellen Evaluierung des Zeitaufwandes des Zugriffsverfahren dargestellt. Die rote Linie gibt den gesamten Zeitaufwand des sequentiellen Abrufens der Einträge aus dem Mapping-Dienst und der Verbindung zu dem DsN an. Die grüne Linie stellt den kumulativen Zeitaufwand des Abrufens der Einträge aus dem Mapping-Dienst, der Verbindung zum DsN und dem sequentiellen Herunterladen der Protokolle dar. Es wurden 500 Versuchsläufe des kompletten Zugriffsverfahrens durchgeführt und dabei wurde jeweils die Dauer gemessen. Auf der x-Achse wird die Nummer des Versuchslaufes dargestellt, auf der y-Achse die Dauer in Millisekunden (ms).

In Tabelle 3.6 werden die Ergebnisse der Evaluierung genauer dargestellt. In der ersten Spalte werden die Schritte, die das Endsystem durchführt, aufgelistet. (1) *Dienst-Abfrage* entspricht der Abfrage der Liste der DsNs für den Dienst. In (2) *DsN-Abfrage* wird das Abrufen der Protokoll-Liste und des Eintrages für das Aufbauen der Virtuellen Verbindung

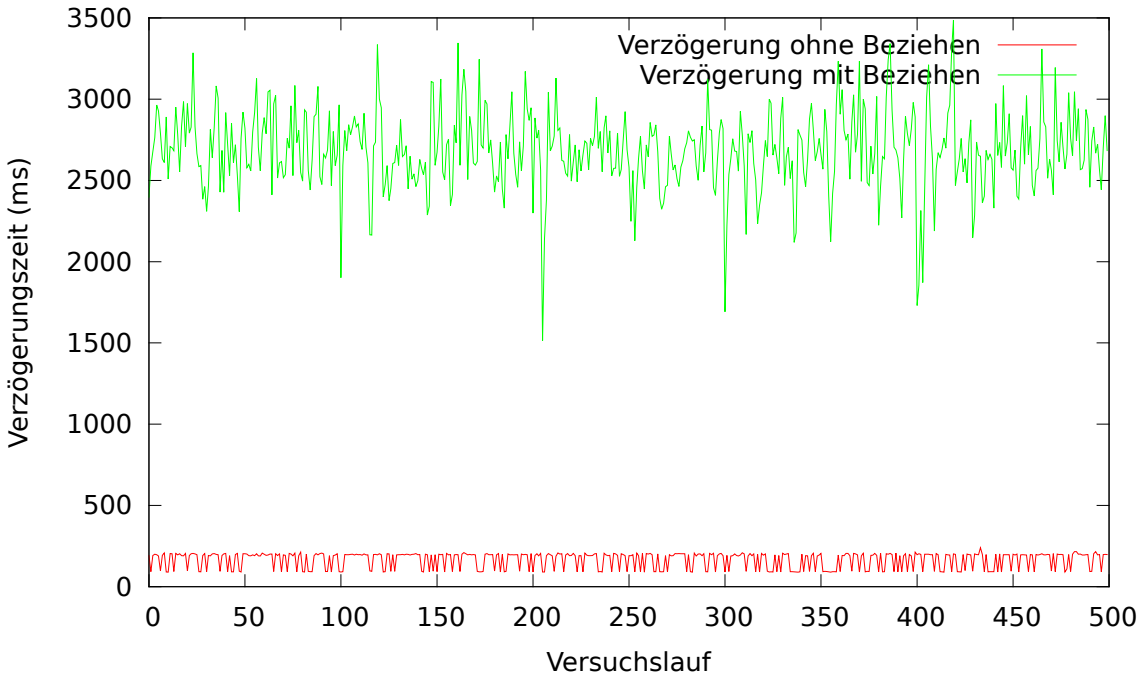

Abbildung 3.10: *Experimentelle Evaluierung des Zeitaufwandes des Zugriffsverfahren für Dienst-spezifische Netze.*

aufgeführt. (3) Die *Protokoll-Abfragen* sind die Abfragen der Einträge für das Beziehen der Protokolle, welches wiederum unter (4) *Protokoll-Beziehen* gelistet wird. In (5) wird das Aufbauen der *Virtuellen Verbindung* gelistet. Unter *Gesamt ohne Beziehen* werden alle Schritte außer das Beziehen der Protokolle zusammengefasst (vgl. rote Linie in Abbildung 3.10). Unter *Gesamt* wird der gesamte Zeitaufwand aufgeführt (vgl. grüne Linie in Abbildung 3.10). In den anderen Spalten werden die minimale, maximale und durchschnittliche (arithmetisches Mittel) Zeit, die für jeden Schritt benötigt wurde, über die 500 Versuchsläufe dargestellt.

Die drei in der Evaluierung heruntergeladenen Protokolle sind Protokolle, die in einem Demonstrator von NENA verwendet wurden. Die Größen der drei Protokolle betragen 344 KB, 224 KB und 552 KB. Damit ergibt sich eine Gesamtgröße von 1120 KB und eine durchschnittliche Geschwindigkeit von 445 Kilobyte/s beim Herunterladen der Protokolle.

Die *Dienst-Abfrage* benötigt im Schnitt 29 ms und die *DsN-Abfrage* 28 ms. Die *Protokoll-Abfragen* benötigen im Durchschnitt 85 ms, da für die drei in dem Experiment verwendeten Dienst-spezifischen Protokolle einzelne Abfragen durchgeführt werden. Die Abfragen aus dem Mapping-Dienst benötigen also im Mittel ca. 28 Millisekunden. Der Zeitaufwand für das Aufbauen der Virtuellen Verbindung liegt im Durchschnitt bei 26 ms. Diese Werte entsprechen im Wesentlichen der Umlaufzeit der Nachrichten zwischen dem Endsystem

Tabelle 3.6: *Details der experimentellen Evaluierung des Zeitaufwandes des Zugriffsverfahren für Dienst-spezifische Netze.*

Aktion	Minimum (ms)	Maximum (ms)	Durchschnitt (ms)
(1) Dienst-Abfrage	13	39	29
(2) DsN-Abfrage	12	47	28
(3) Protokoll-Abfragen	37	112	85
(4) Protokoll-Beziehen	1424	3285	2518
(5) Virtuelle Verbindung	24	65	26
Gesamt ohne Beziehen	89	239	166
Gesamt	1514	3487	2684

und den entsprechenden Netzkomponenten. Der Zeitaufwand für alle diese Schritte liegt im Mittel bei 166 ms. Die Schwankungen der Verzögerungszeit zwischen zwei Werten (siehe rote Linie in Abbildung 3.10) ergeben sich dadurch, dass die Umlaufzeit der Abfragen aus dem Mapping-Dienst aus dem Cluster Darmstadt etwa halb so lang wie aus den beiden anderen Clustern ist. Das Beziehen der Protokolle benötigt im Durchschnitt 2518 ms und wird im Wesentlichen durch die Geschwindigkeit der Datenübertragung und die Größe der Protokolle bestimmt. Das Beziehen der Protokolle ist um ein Vielfaches langsamer als die Abfragen im Mapping-Dienst und verursacht damit den größten Zeitaufwand. Der gesamte Zeitaufwand des Zugriffsverfahrens beträgt demnach in diesem Experiment durchschnittlich 2684 ms. Damit benötigt das Zugriffsverfahren weniger als drei Sekunden, um auf ein DsN zuzugreifen.

Die Ergebnisse dieser Evaluierung zeigen, dass der Zeitaufwand der Abfragen im Mapping-System und des Aufbauens der Virtuellen Verbindung gering ist. Die für diese Schritte benötigte Zeit wird im Wesentlichen von der Umlaufzeit der Nachrichten zwischen dem Endsystem und den entsprechenden Netzkomponenten bestimmt. Die gesamte Dauer des Zugriffsverfahrens wird von der für das Herunterladen der Protokolle benötigten Zeit dominiert. Dennoch liegt die gesamte Dauer unter drei Sekunden. Diese Dauer ist für den erstmaligen Zugriff auf ein DsN akzeptabel. Solange die Verbindung zu dem DsN nicht abgebaut wird (z.B. auf Wunsch des Nutzers, nach Ablauf eines Timers bei Inaktivität, oder aus Ressourcenmangel auf dem Endsystem), erfolgen alle späteren Zugriffe auf dieses Netz ohne die zusätzliche Verzögerung des Zugriffsverfahrens. In vielen Fällen, besonders wenn dem Nutzer ein Auswahl-Fenster für die DsNs präsentiert wird, ist die durch das Zugriffsverfahren verursachte Verzögerung kaum bemerkbar. Für Fälle, in denen eine solche Verzögerung nicht akzeptabel ist, gibt es verschiedene Möglichkeiten, den Zugriff zu beschleunigen. So können beispielsweise die Abfragen im Mapping-Dienst und das Herunterladen der Protokolle parallelisiert werden. Außerdem

ist es möglich, Verbindungen zu beliebten DsNs bereits beim Starten des Endsystems oder entsprechender Anwendungen aufzubauen und nicht auf die Anfrage des Nutzers zu warten. Darüber hinaus ist es möglich, die Dienst-spezifische Protokolle häufig verwendeter DsNs in einem persistenten Speicher zu hinterlegen (Caching), damit die Dienst-spezifischen Protokolle z.B. nach einem Neustart des Endsystems nicht erneut bezogen werden müssen.

3.9 Zusammenfassung

In diesem Kapitel wurde EUNA, ein flexibles Zugriffsverfahren für DsNs vorgestellt. Das Zugriffsverfahren erlaubt Endsystemen, transparent für den Nutzer DsNs für Dienste zu finden, die Dienst-spezifischen Protokolle von DsNs zu beziehen und Virtuelle Verbindungen zu DsNs aufzubauen. Das Zugriffsverfahren erreicht Flexibilität dadurch, dass sowohl die Methoden für das Beziehen der Dienst-spezifischen Protokolle als auch die Methoden für das Aufbauen Virtueller Verbindungen ausgetauscht werden können. Außerdem erlaubt das Zugriffsverfahren den Dienste-Anbietern, festzulegen, wie Dienst-spezifische Protokolle bezogen und Virtuelle Verbindungen aufgebaut werden können. Endsysteme können aus den Vorgaben des Dienste-Anbieters basierend auf den verfügbaren Methoden geeignete Bezugsmethoden und Verbindungsmethoden auswählen. Anhand eines Prototypen wurde die Funktionsfähigkeit des Zugriffsverfahren gezeigt und der beim Zugriff auf Dienste verursachte Zeitaufwand evaluiert. Dabei wurde gezeigt, dass der Zeitaufwand beim Zugriff auf ein beispielhaftes DsN nur wenige Sekunden beträgt.

Kapitel 4

Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen

Unterhält ein Endsystem mehrere Verbindungen zu DsNs, teilen diese sich die begrenzten CPU-, Speicher- und Netz-Ressourcen des Endsystems. Um die Anforderungen verschiedener, parallel verwendeter Dienste einhalten zu können, wird eine Möglichkeit benötigt, um den gegenseitigen Einfluss zu minimieren. Wenn beispielsweise ein Protokoll eines DsN aufwändige Berechnungen auf den übertragenen Daten ausführt, könnte hierdurch die Verarbeitung eines zeitkritischen Protokolls eines anderen DsN verzögert werden. Es ist also nötig, den Ressourcen-Verbrauch von DsNs zu regulieren. Eine besondere Herausforderung bei der Verwaltung von Ressourcen ist die Dynamik durch neu hinzukommende und wegfallende Verbindungen zu DsNs sowie die Heterogenität der Eigenschaften der DsNs und Dienst-spezifischen Protokolle. Ein Endsystem besitzt kein Vorwissen über die Verbindungen, die zur Laufzeit zu DsNs aufgebaut werden, und kein Wissen über die Protokolle, die in den DsNs verwendet werden. In diesem Kapitel wird das Hierarchische Knoten-Management (HKM), ein flexibles Verwaltungssystem für Endsysteme vorgestellt. Es erlaubt, die auf einem Endsystem verfügbaren Ressourcen auf verschiedene DsNs und die darin verwendeten Protokolle aufzuteilen und dabei Nutzer-Richtlinien zu beachten. Das Verwaltungssystem besteht aus mehreren Verwaltungskomponenten, die in einer Baum-förmigen Hierarchie organisiert sind. Dabei bildet eine für das Endsystem zentrale Knoten-weite Verwaltungskomponente die Wurzel des Baumes. Auf der Baum-Ebene darunter befindet sich für jede Verbindung zu einem DsN eine Netz-spezifische Verwaltungskomponente, die an das DsN angepasst ist. Auf der Blatt-Ebene des Baumes befindet sich schließlich für jedes Dienst-spezifische Protokoll eine Protokoll-spezifische Verwaltungskomponente, die an das entsprechende Protokoll angepasst ist. Die Protokoll-spezifischen Verwaltungskomponenten, die zu Dienst-spezifischen Protokollen des selben DsN gehören, sind der selben Netz-spezifischen Verwaltungskomponente zugeordnet. Dies ermöglicht der Netz-spezifischen Verwaltungskomponente

eine Protokoll-übergreifende Verwaltung. Jede Verwaltungskomponente führt einen Management-Zyklus bestehend aus Monitoring, Entscheidungsfindung und Anpassung durch. Entlang der Hierarchie können sich die Verwaltungskomponenten Ressourcen zuweisen oder entziehen. Auf die Veränderungen der ihnen zugewiesenen Ressourcen können die Verwaltungskomponenten individuell reagieren. Dank der Hierarchie wird ein hoher Grad an Flexibilität erreicht. Neue Verwaltungskomponenten können hinzugefügt und bestehende entfernt werden, ohne das gesamte Verwaltungssystem anpassen zu müssen. Detailliertes Wissen über die Verwaltung der DsNs oder der Dienst-spezifischen Protokolle wird nur in den entsprechenden Verwaltungskomponenten benötigt. Außerdem müssen nur die Schnittstellen zwischen der Knoten-weiten Verwaltungskomponente und den Netz-spezifischen Verwaltungskomponenten festgelegt werden. Zwischen den Netz-spezifischen Verwaltungskomponenten und den ihnen zugeordneten Protokoll-spezifischen Verwaltungskomponenten können individuelle, an die entsprechenden DsNs angepasste Schnittstellen verwendet werden.

Gliederung

Dieses Kapitel ist folgendermaßen gegliedert. In Abschnitt 4.1 wird näher auf die Problemstellung eingegangen. Dabei wird gezeigt, dass ein flexibles Verwaltungssystem für DsNs benötigt wird. In Abschnitt 4.2 wird der Stand der Technik vorgestellt. In Abschnitt 4.3 wird das Hierarchische Knoten-Management präsentiert. In den darauf folgenden Abschnitten wird das HKM näher betrachtet. In Abschnitt 4.4 wird auf die Überwachung von DsNs eingegangen. In Abschnitt 4.5 werden die Interaktion mit dem Nutzer und Richtlinien präsentiert. In Abschnitt 4.6 wird gezeigt, wie das Hierarchische Knoten-Management die Ressourcen eines Endsystems verwaltet. In Abschnitt 4.7 wird die Implementierung des HKM als Prototyp vorgestellt. In Abschnitt 4.8 wird das HKM mit Hilfe des Prototypen evaluiert. In Abschnitt 4.9 werden die Ergebnisse dieses Kapitels zusammengefasst.

4.1 Problemstellung

Für die Nutzung von DsNs wird eine flexible Ressourcen-Verwaltung gebraucht. Die Verwaltung muss mit der durch DsNs verursachten Dynamik auf einem Endsystem sowie den heterogenen Eigenschaften von DsNs und Dienst-spezifischen Protokollen umgehen können. Die auf dem Endsystem verwendeten Dienst-spezifischen Protokolle und die Virtuellen Verbindungen zu DsNs ändern sich zur Laufzeit durch das Verhalten des Nutzers. Diese Problemstellung wird in diesem Abschnitt anhand des Beispiels in Abbildung 4.1 verdeutlicht.

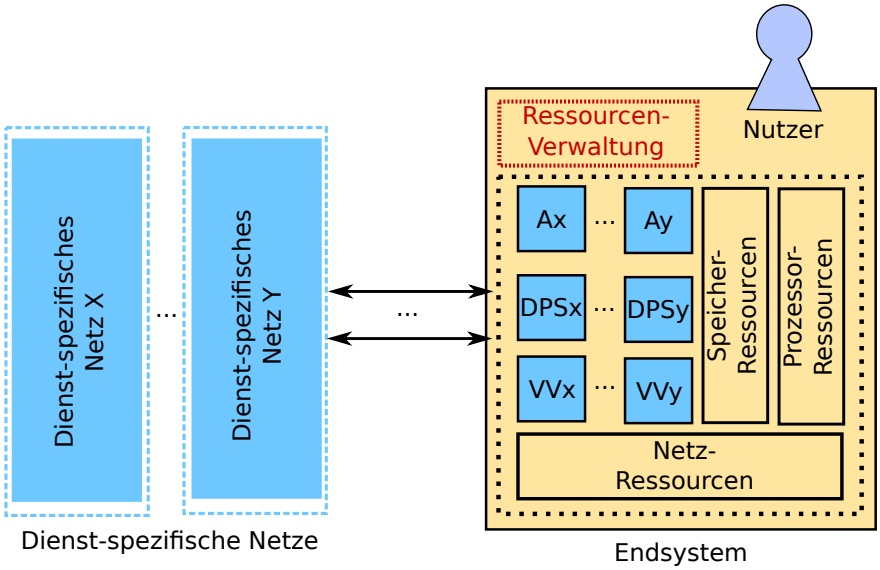

Abbildung 4.1: *Verwaltung der Netz-, Speicher- und Prozessor Ressourcen eines Endsystems*

In Abbildung 4.1 werden der Zugriff von einem Endsystem auf DsNs und die Ressourcen des Endsystems schematisch dargestellt. Ein Nutzer auf einem Endsystem verwendet verschiedene Anwendungen *Ax* bis *Ay*, um auf Dienste in den DsNs *X* bis *Y* zuzugreifen. Für den Zugriff auf die DsNs betreibt das Endsystem Dienst-spezifische Protokolle in den Dienst-spezifischen Protokoll-Stapeln *DPSx* bis *DPSy*. Außerdem betreibt das Endsystem für die Anbindung an die DsNs die Virtuellen Verbindungen *VVx* bis *VVy*. Das Endsystem besitzt beschränkte Netz-, Speicher- und Prozessor-Ressourcen. Die Anwendungen, Dienst-spezifischen Protokolle und die Virtuellen Verbindungen, die auf dem Endsystem betrieben werden, verwenden und konkurrieren um diese Ressourcen. Dadurch können sich die Dienst-spezifischen Protokolle und Virtuellen Verbindungen gegenseitig beeinflussen. Im Folgenden werden drei Beispiele aufgeführt, die dies verdeutlichen.

(1) Beispielsweise kann eine Virtuelle Verbindung Verschlüsselung einsetzen, um die übertragenen Daten vor Dritten zu schützen. Dadurch führt sie eventuell komplexe Operationen auf den Paketen aus, die viel Prozessor-Zeit des Prozessors verbrauchen. Wird ein Dienst-spezifisches Protokoll für eine Zeit-kritische Anwendung auf dem selben Prozessor bzw. Prozessor-Kern ausgeführt, kann dessen Ausführung durch die Virtuelle Verbindung verzögert werden, wodurch das Dienst-spezifische Protokoll eventuell seine Zeit-Schranken nicht einhalten kann.

(2) Ein weiteres Beispiel besteht aus zwei Dienst-spezifischen Protokollen auf einem Endsystem. Eines wird verwendet, um eine Datei herunter zu laden. Das andere wird verwendet, um ein Video über einen Video-Streaming Dienst zu empfangen. Durch das Herunterladen kann ein großer Teil der möglichen Datenrate der Netzwerkschnittstelle des Knotens belegt werden. Hierdurch kann die noch verfügbare Datenrate nicht ausrei-

chen, um das Video mit einer ausreichenden Datenrate zu empfangen und in einer für den Nutzer ausreichenden Qualität abzuspielen.

(3) Ein anderes Beispiel besteht wiederum aus zwei Dienst-spezifischen Protokollen. Eines (z.B. ein CCN-Protokoll [55]) betreibt einen großen Cache, um empfangene Dateneinheiten für lokale Anwendungen oder andere Knoten zwischenzuspeichern. Das andere erstellt eigene Nachrichten für den Austausch von Kontroll-Informationen mit weiteren Knoten. Belegt das erste Protokoll zu viel Speicher des Endsystems für das Zwischenspeichern von Nachrichten, kann für das andere Protokoll der Speicher nicht ausreichen, um Kontroll-Nachrichten zu erstellen. Hierdurch kann dessen Funktionsfähigkeit eingeschränkt oder sogar verhindert werden.

Die drei Beispiele zeigen, dass auf einem Endsystem geeignete Mechanismen benötigt werden, um die Ressourcen des Endsystems zu verwalten. Ein Verwaltungssystem sollte dem Endsystem ermöglichen, den Ressourcen-Verbrauch von Dienst-spezifischen Protokollen und Virtuellen Verbindungen zu regulieren. Im Folgenden werden die auf einem Endsystem verfügbaren Ressourcen näher vorgestellt. Anschließend wird auf die Ressourcen-Verwaltung und ihre Besonderheiten in DsNs eingegangen.

Ressourcen

Prozessor-Ressourcen: Unter den Prozessor-Ressourcen eines Endsystems versteht man die verfügbare Prozessor-Zeit eines Prozessors und die verfügbaren Prozessor-Kerne des Endsystems. Werden Dienst-spezifische Protokolle oder Virtuelle Verbindungen auf einem Prozessor ausgeführt, verbrauchen sie die Prozessor-Zeit des entsprechenden Prozessors. Besitzt der Prozessor des Endsystem mehrere Prozessor-Kerne, belegen Dienst-spezifische Protokolle oder Virtuelle Verbindungen alle oder einen Teil der verfügbaren Prozessor-Kerne bei ihrer Ausführung.

Speicher-Ressourcen: Unter den Speicher-Ressourcen versteht man den verfügbaren Speicher, sowohl fest als auch flüchtig, des Endsystems. Für den festen Speicher wird Festplatten-Speicher angenommen. Für den flüchtigen Speicher wird der Arbeitsspeicher des Endsystems angenommen. Empfangen oder versenden Dienst-spezifische Protokolle oder Virtuelle Verbindungen Nachrichten, belegen diese Nachrichten Speicher auf dem Endsystem. Außerdem können Nachrichten gepuffert oder Zustände gespeichert werden, die wiederum Speicher verbrauchen.

Netz-Ressourcen: Unter den Netz-Ressourcen eines Endsystems versteht man die mögliche Datenrate einer Netzwerkschnittstelle und die verschiedenen Netzwerkschnittstellen des Endsystems. Tauschen Dienst-spezifische Protokolle oder Virtuelle Verbindungen Dateneinheiten über eine Netzwerkschnittstelle aus, belegen sie diese für die Dauer der Übertragung. Tauschen mehrere Dienst-spezifische Protokolle oder Virtuelle Verbindungen im selben Zeitraum Daten über eine Netzwerkschnittstelle aus, teilen sich diese die mögliche Datenrate der Netzwerkschnittstelle.

Ressourcen-Verwaltung

Es wird ein Verwaltungssystem benötigt, das Endsystemen die Zuteilung der lokal verfügbaren Ressourcen auf die von dem Endsystem verwendeten Komponenten der DsNs erlaubt. Hierfür muss das Verwaltungssystem die Ressourcen den Virtuellen Verbindungen und den Dienst-spezifischen Protokoll-Stapeln der DsNs, auf die das Endsystem zugreift, zuteilen. Außerdem müssen die Ressourcen den Dienst-spezifischen Protokollen innerhalb der Dienst-spezifischen Protokoll-Stapel zugeteilt werden können. Das Verwaltungssystem muss eine ständige Anpassungen der Ressourcen-Zuteilungen zur Laufzeit erlauben. Der Verbrauch der Netz-, Speicher- und Prozessor-Ressourcen der Dienst-spezifischen Protokolle und Virtuellen Verbindungen kann sich zur Laufzeit ändern. Beispiele hierfür sind (1) Anpassungen der Dienst-spezifischen Protokolle an die Eigenschaften der Netz-Infrastruktur durch Fehler-Korrektur-Mechanismen, (2) Puffer-Anpassungen oder (3) Verarbeitung von Daten in den Paketen.

(1) Ein Beispiel für Fehler-Korrektur-Mechanismen sind Forward Error Correction (FEC) Mechanismen, die erlauben durch zusätzliche Redundanzen Fehler bei der Übertragung zu korrigieren. Werden die Redundanzen zur Laufzeit an die Fehlerwahrscheinlichkeit der Übertragung über die Netz-Infrastruktur angepasst, ändern sich damit auch die benötigten Netz-Ressourcen zur Laufzeit.

(2) Ein Beispiel für die Anpassung von Puffer-Größen sind die Puffer, die zum Abspielen von empfangenen Video-Daten mit konstanter Verzögerung verwendet werden. Schwanken bei der Übertragung von Video-Daten die Verzögerungen und die Datenraten, kann ein solcher Puffer eine fehlerfreie Wiedergabe eines Videos ermöglichen. Passt ein solches Verfahren die verwendete Puffer-Größe an die aktuellen Verzögerungszeiten bzw. Datenraten zur Laufzeit an, ändern sich damit auch die benötigten Speicher-Ressourcen zur Laufzeit.

(3) Ein Beispiel für die Verarbeitung der Daten innerhalb von Paketen ist das Komprimieren von Daten in Übertragungsprotokollen. Dies erlaubt die Größe der übertragenen Pakete bei der Übertragung so anzupassen, dass nicht die verfügbare Datenrate in der Netz-Infrastruktur, die durch die Übertragungsprotokolle ermittelt wurde, überschritten wird. Hierfür komprimiert der Sender die Daten und dementsprechend müssen beim Empfang der Pakete die entsprechenden Daten wieder dekomprimiert werden. Passt der Sender zur Laufzeit die Stärke der Komprimierung an die verfügbare Datenrate an, ändern sich damit auch die benötigten Prozessor-Ressourcen zur Laufzeit.

Konkurrieren mehrere DsNs oder Dienst-spezifische Protokolle um gemeinsame Ressourcen, kann es außerdem sinnvoll sein, Prioritäten vergeben zu können. Bezieht beispielsweise ein Nutzer, wie im weiter oben beschriebenen Beispiel, gleichzeitig ein Video von einem Video-Streaming Dienst und lädt eine große Datei herunter, konkurrieren diese Übertragungen um die verfügbare Datenrate des Endsystems. Ist das Video dem Nutzer wichtiger, sollten für dessen Übertragung die benötigten Ressourcen bereit gestellt

werden, auch wenn dies die Datenrate für das Herunterladen der Datei reduziert. Daher sollte das Verwaltungssystem Nutzer-Richtlinien unterstützen, die dem Nutzer erlauben, zu spezifizieren, wenn eine Anwendung bzw. ein Dienst wichtiger ist als andere.

Die hier beschriebenen Anpassungen der Ressourcen an die Netz-Eigenschaften erfordern eine Überwachung (Monitoring) von Dienst-spezifischen Protokollen, Virtuellen Verbindungen und der Netz-Eigenschaften zur Laufzeit. Außerdem erfordern sie Management-Entscheidungen, wie auf die sich ändernden Netz-Eigenschaften durch Veränderung der Parameter von Dienst-spezifischen Protokollen oder Virtuellen Verbindungen reagiert werden muss. Das Verwaltungssystem muss ein solches Monitoring und solche Management-Entscheidungen unterstützen.

Besonderheiten

Eine Besonderheit von DsNs, die sich auf die Ressourcen-Verwaltung auswirkt, ist die Heterogenität der DsNs und Dienst-spezifischen Protokolle sowie die Dynamik auf dem Endsystem. Greift der Nutzer auf verschiedene Dienste zu, greift das Endsystem auf die entsprechenden DsNs zu, die die Dienste anbieten. Die DsNs und Dienst-spezifischen Protokolle können heterogene Eigenschaften besitzen und unterschiedliche Mechanismen zur Verwaltung bieten. Beispielsweise könnte ein DsN für ein CCN auf hohe Datenraten ausgelegt sein. Die Dienst-spezifischen Protokolle könnten Mechanismen bieten, die die Größe des Caches für Inhalte regulieren. Ein DsN für Video-Telefonie könnte für geringe Latenzen und geringen Jitter optimiert sein. Die darin verwendeten Dienst-spezifischen Protokolle könnten Forward-Error-Correction Mechanismen verwenden, die eine Steuerung der übertragenen Redundanzen erlauben. In einem DsN, das für eine sichere Kommunikation entworfen ist, könnten Dienst-spezifische Protokolle betrieben werden, die eine Parametrisierung der verwendeten Verschlüsselungsverfahren erlauben. Es ist nicht absehbar, welche Eigenschaften zukünftige DsNs bieten und welche Mechanismen zur Verwaltung sie besitzen. Ein Verwaltungssystem sollte daher flexibel sein, um zukünftige Entwicklungen zu unterstützen.

Die Dynamik auf dem Endsystem wird dadurch verursacht, dass das Endsystem für jedes DsN, auf das es zugreift, eine Virtuelle Verbindung und Dienst-spezifische Protokolle betreibt. Diese Menge von Virtuellen Verbindungen und Dienst-spezifischen Protokollen ändert sich zur Laufzeit durch das Nutzer-Verhalten. Greift der Nutzer auf neue Dienste zu, kommen neue DsNs hinzu. Werden Dienste nicht mehr genutzt, können Verbindungen zu DsNs abgebaut und die entsprechenden Dienst-spezifischen Protokolle entfernt werden. Daher muss das Verwaltungssystem flexibel sein, um mit der sich zur Laufzeit ändernden Menge von DsNs umgehen zu können.

4.2 Stand der Technik

Es existiert eine Vielzahl verschiedener Verwaltungssysteme, die unterschiedliche Management-Aufgaben des FCAPS-Prinzip [54] abdecken. Einfache Verwaltungssysteme wie SNMP [43] erlauben beispielsweise die Konfiguration von Komponenten und basieren darauf, dass ein Nutzer explizit Management-Aufgaben an bestimmte Komponenten übermittelt. Modernere Verwaltungssysteme erlauben ein autonomes Management von Komponenten oder Netzen. Hierfür setzen sie einen autonomen Management-Zyklus, wie in [35, 59] beschrieben, ein. Vereinfacht besteht dieser aus Monitoring, Entscheidungsfindung und der Ausführung von Entscheidungen. Über Monitoring werden Informationen über Komponenten oder das Netz erlangt. Basierend auf diesen Informationen wird eine Management-Entscheidung getroffen und ausgeführt, worauf der Zyklus von Neuem beginnt. Diese Verwaltungssysteme wurden jeweils für eine einzelne Netzwerk-Architektur entworfen. DsNs erlauben allerdings die Verwendung verschiedener Netzwerk-Architekturen auf einem Endsystem zur selben Zeit. Die auf dem Endsystem verwendeten DsNs und Protokolle können daher unterschiedlichste Eigenschaften und Anforderungen besitzen. Daher wurde das in diesem Kapitel vorgestellte Verwaltungssystem für Endsysteme in DsNs entworfen, das eine Verwaltung der Ressourcen eines Endsystems ermöglicht. Außerdem wurde NENA [74], ein Rahmenwerk für DsNs um dieses Verwaltungssystem erweitert.

Im Folgenden werden Verwaltungssysteme für zukünftige Netze vorgestellt, die relevant für diese Arbeit sind. Anschließend werden für die Verwaltung von Endsystemen in DsNs wichtige Aspekte behandelt und die Erkenntnisse dieses Abschnittes zusammengefasst.

4.2.1 Management in zukünftigen Netzen

Einige existierende Forschungsarbeiten im Bereich zukünftiger Netze beschäftigen sich mit dem Thema Management. Beispiele solcher Arbeiten werden im Folgenden vorgestellt.

Im Rahmen des 4WARD-Projektes [1] wurden die Konzepte des so genannten *In-Network Management* erarbeitet und in einigen Publikationen beschrieben [36, 97, 31]. Das In-Network Management besteht aus *Self Managed Entities* (SME), *Global Management Points* (GMP) und *Management Domains*. SMEs sind selbstständige Verwaltungskomponenten, die im Netz verteilt sind. Das In-Network Management setzt mit den SMEs die Prinzipien *Co-Design* und *Co-Location* um. Dies bedeutet, SMEs werden explizit für die Komponenten, die sie verwalten, entworfen und mit den Komponenten, die sie verwalten, gekoppelt bzw. im Netz ausgebracht. SMEs werden in Management Domains organisiert, um gemeinsam Management-Aufgaben zu erfüllen. Eine Management Domain wird über einen zentralen Global Management Point verwaltet. Der GMP bietet

eine Schnittstelle nach außen, über die der Management Domain Management-Aufträge erteilt oder Richtlinien wie zum Beispiel Service Level Agreements (SLAs) übergeben werden können. Der GMP setzt Management-Aufträge und Richtlinien um, indem er sie weiterverarbeitet und an SMEs weiterreicht.

Im Rahmen der FIRE Initiative [101] wurde im Projekt *SelfNET* [108] ein autonomes Management für das zukünftige Internet untersucht. In *Future Internet Elements* [62] wird ein Management-Ansatz vorgeschlagen, der aus autonomen Verwaltungskomponenten besteht, die in einer Hierarchie organisiert sind. Die Verwaltungskomponenten sind Kontext-bewusst, d.h. sie sammeln durch Monitoring Wissen über sich selbst und ihre Umgebung. Die unterste Hierarchie-Ebene besteht aus den einzelnen Knoten, die mittlere Ebene besteht aus Netz-Domänen und die oberste Ebene beschreibt den Einflussbereich eines Anbieters bzw. Netz-Betreibers. Auf jeder Hierarchie-Ebene wird eine verteilte Variante des Management-Zyklus bestehend aus Monitoring, Entscheidungsfindung und Ausführung der Management-Entscheidungen durchgeführt. Das Ziel dieser Aufteilung ist das Erreichen eines hohen Grades an Autonomie und schnelle lokale Entscheidungsfindungen und Ausführungen.

In *CogNET* [92] wird eine Architektur für kognitive drahtlose Netze vorgestellt, die Kontroll- und Daten-Ebene voneinander trennt. Einzelne Knoten sammeln mit Monitoring Informationen über sich selbst und ihre Umgebung. Die Knoten sind über eine separate Schnittstelle an eine globale Kontroll-Ebene angeschlossen, die *Global Control Plane* (GCP), die als ein Overlay-Netz realisiert ist. Knoten können andere Knoten über die GCP finden und ihre über das Monitoring gesammelten Informationen austauschen. Dies erlaubt den Knoten die Daten-Ebene zur Laufzeit an den Netz-Zustand anzupassen, indem beispielsweise die Funk-Schnittstelle des Knotens konfiguriert oder das bei der Übertragung zwischen zwei Knoten verwendete Frequenz-Spektrum angepasst wird.

In der Veröffentlichung *Autonomous Decentralized Management Architecture for MANETs* (ADMA) [4] wird eine autonome, dezentralisierte Management-Architektur für Mobile Ad-hoc Netze (MANETs) vorgestellt. Die Architektur besteht aus vier Komponenten auf einem Knoten, die den Management-Zyklus umsetzen. Außerdem werden Management-Ziele durch Richtlinien bestimmt. Eine Monitoring-Komponente sammelt Informationen über den Zustand des Knotens. Der *Local Policy Decision Point* (LPDP) trifft autonome Management-Entscheidungen auf dem Knoten basierend auf dem Zustand und entsprechenden Richtlinien. Der *Policy Enforcement Point* (PEP) setzt die Management-Entscheidungen des LPDP auf dem Knoten um. In dem *Local Policy Repository* werden die Richtlinien gespeichert. ADMA erlaubt, das Netz an sich ändernde Eigenschaften anzupassen ohne Interaktion mit Menschen und ohne zentrale Instanzen.

In *Services Integration, controL, and Optimization* (SILO) [25] wird eine Architektur für zukünftige Netze vorgestellt. Auf einem Knoten werden Kommunikationsdienste, angepasst an die Anforderungen des Nutzers und die Netz-Eigenschaften, bei Bedarf und zur Laufzeit erstellt. Die Dienste werden dabei von einem *Control Agent* aus Bausteinen,

die Funktionalitäten enthalten, zusammengesetzt und als so genannte SILOs geladen. Die Bausteine besitzen Schnittstellen, so genannte *Knobs*, über die sie zur Laufzeit angepasst werden können. Somit ist es möglich zum Beispiel die Kompressionsrate in einem Kompressionsbaustein zur Laufzeit anzupassen. Der Control Agent kann also mit Hilfe der Knobs ein Management der Kommunikationsdienste auf dem Knoten realisieren. Darüber hinaus könnten Management- und Monitoring Funktionalitäten in eigenen Bausteinen realisiert und in SILOs bei deren Erstellung integriert werden.

Im Rahmen des Projektes *Autonomic Network Architecture* (ANA) [3] wurde eine Architektur für zukünftige Netze entworfen [57]. Diese Architektur basiert auf Funktionalen Blöcken, die zur Laufzeit zusammengesetzt werden, um Kommunikationsdienste zu erstellen. In [75] werden Anforderungen an das Monitoring in zukünftigen Netzen behandelt und die darauf aufbauende Monitoring-Lösung in der ANA-Architektur vorgestellt. Da Kommunikationsdienste zur Laufzeit aus Funktionalen Blöcken zusammengesetzt werden, existiert kein a priori Wissen darüber, was wie überwacht werden kann. Das Monitoring muss zur Laufzeit die Umgebung, in der es aktiv ist, ermitteln und die Monitoring-Komponenten selbst müssen so zusammengesetzt werden können, dass ein dynamisches, verteiltes Monitoring möglich ist. Hierfür verwendet die Monitoring-Lösung so genannte *Information Hooks*. Ein Information Hook ist eine selbst-beschreibende Schnittstelle. Eine solche Schnittstelle stellt eine Beschreibung bereit, welche Informationen über die Schnittstelle bezogen werden können. Jeder Funktionale Block in ANA besitzt eine solche Schnittstelle. Ein Funktionaler Block für das Monitoring (MFB) stellt über seinen Information Hook eine Beschreibung dessen bereit, was durch den MFB überwacht wird, z.B. Latenz oder Fehlerrate. Außerdem werden verschiedene MFBs in so genannten *Monitoring Compartments* organisiert. Ein Monitoring Compartment ist eine Gruppe von kooperierenden MFBs. MFBs können mit Hilfe der Information Hooks und der Monitoring Compartments Informationen austauschen und somit ein verteiltes Monitoring realisieren.

4.2.2 Monitoring

Ein Verwaltungssystem benötigt Informationen über den Zustand von Knoten oder des Netzes, um Management-Entscheidungen zu treffen. Diese Informationen werden durch Monitoring erlangt. Das Monitoring benötigt festgelegte Schnittstellen oder Wissen über die Details der überwachten Komponenten. In [75] wird die Problematik von fehlendem a priori Wissen beim Monitoring diskutiert. Diese Problematik trifft auch auf die Verwaltung von DsNs auf einem Endsystem zu. Bei DsNs werden verschiedene Komponenten auf einem Endsystem ausgeführt, über deren Details kein Wissen vorliegt. Eine Besonderheit bei DsNs ist, dass Dienst-spezifische Protokolle jeweils für das DsN entworfen werden, in dem sie betrieben werden. Hierdurch können beim Entwurf von Verwaltungskomponenten für die DsNs, wie bei INM [36], die Konzepte Co-Design und Co-Location verwendet

werden. Das in diesem Kapitel vorgestellte Verwaltungssystem verwendet daher eine Kombination der Konzepte aus [75] und [36]. Verwaltungskomponenten werden für DsNs und Dienst-spezifische Protokolle entworfen. Hierdurch besitzen sie Detail-Wissen über die entsprechenden Komponenten. Der Austausch von Informationen zwischen Verwaltungskomponenten verläuft mit Hilfe von einfachen Schnittstellen.

Für eine effiziente Verwaltung wird ein Austausch von Informationen zwischen Komponenten und Netz-Knoten benötigt. Hierdurch kann beispielsweise ein Knoten Informationen über den Zustand des Netzes erhalten. Wird dabei z.B. ein Ausfall eines Knotens erkannt, kann das Routing angepasst werden oder bei der Kommunikation auf andere Knoten ausgewichen werden. Für den Austausch von Informationen können Unicast Protokolle wie SNMP [43] eingesetzt werden oder Gruppen-Kommunikation wie Gossiping-basierte [125, 126] und Baum-basierte [127] Ansätze verwendet werden. In DsNs ist keine allgemeingültige Lösung möglich, die in allen DsNs ideal funktioniert. Daher erlaubt das in diesem Kapitel beschriebenen Verwaltungssystem jedem DsN, eigene Mechanismen zu implementieren.

4.2.3 Entscheidungsfindung und Ausführung

In INM [36] und *Autonomic and Decentralized Management of Wireless Access Networks* [100] wird gezeigt, dass die Entscheidungsfindung und damit die Ausführung von Management-Aufgaben in einem autonomen Verwaltungssystem beeinflussbar sein muss. Autonome Management-Entscheidungen können zu Zyklen bzw. zyklischen Management-Entscheidungen führen. Daher sollte ein autonomes Verwaltungssystem eine direkte oder indirekte Beeinflussung der Management-Entscheidungen ermöglichen. Bei der direkten Beeinflussung wird der Nutzer direkt in die Entscheidungsfindung integriert bzw. kann diese direkt beeinflussen. Dies resultiert in einem höheren Aufwand für den Nutzer. Bei der indirekten Beeinflussung beeinflusst der Nutzer nur indirekt die Management-Entscheidungen durch Richtlinien. Richtlinien werden von Nutzer festgelegt, vom Verwaltungssystem interpretiert und bei den Management-Entscheidungen, bei denen der Nutzer nicht mehr direkt beteiligt ist, beachtet bzw. umgesetzt. Hierdurch wird ein geringerer Grad an Interaktion mit Nutzer als beim direkten Beeinflussen benötigt. Außerdem muss der Nutzer kein (oder zumindest wesentlich weniger als bei der direkten Beeinflussung) Wissen über Netz-Details besitzen.

Da bei DsNs der Nutzer auf beliebige DsNs zugreift, soll er kein Wissen über deren Details benötigen. Daher erlaubt das in diesem Kapitel beschriebene Verwaltungssystem eine indirekte Beeinflussung durch Richtlinien. Wie beim Monitoring, sollte die Umsetzung der Management-Entscheidung und eine eventuelle Signalisierung von Management-Entscheidungen zwischen Komponenten individuell an die Anforderungen von DsNs angepasst werden. Daher erlaubt das in diesem Kapitel beschriebene Verwaltungssystem jedem DsN, eigene Mechanismen zu implementieren.

4.2.4 Zusammenfassung

In der SILO Netzwerk-Architektur [25] werden Schnittstellen zum Management und zur Optimierung von Komponenten verwendet. Diese Schnittstellen (Knobs) erlauben, die Parameter der Komponenten zur Laufzeit zu verändern. ANA [57] verwendet ähnliche Schnittstellen, die auch genutzt werden können, um Monitoring zu realisieren. Zusätzlich schlägt ANA eine Netzwerk-Architektur vor, die ermöglicht, Komponenten in Gruppen zusammenzufassen. In einer solchen Gruppe können die Komponenten zusammen arbeiten, um Management-Aufgaben zu lösen. Durch Monitoring gesammelte Informationen können ausgetauscht, zusammengefasst und an interessierte Komponenten weitergereicht werden. Das In-Network Management (INM) [36] erlaubt wie ANA, Komponenten in Gruppen zusammenzufassen. Dadurch wird ein gemeinsames Management der gruppierten Komponenten erreicht. Darüber hinaus schlägt INM die Konzepte Co-Design (Komponenten zusammen mit ihren Management-Funktionalitäten entwerfen) und Co-Location (Komponenten mit ihren Management-Funktionalitäten bündeln) für das Management vor. In CogNET [92] tauschen verteilte Komponenten über eine gemeinsame Kontroll-Ebene Informationen aus. In SelfNET [62] werden Komponenten in einer mehrstufigen Hierarchie angeordnet, wobei auf jeder Hierarchie-Ebene ein Management-Zyklus durchgeführt wird. In ADMA [4] und INM [36] werden Richtlinien verwendet, um das Management zu beeinflussen.

Das Hierarchische Knoten-Management, das in diesem Kapitel vorgestellt wird, ist an die Verwendung von DsNs angepasst. Dienst-spezifische Protokolle werden zusammen mit ihren Verwaltungskomponenten entworfen. Außerdem werden sie mit ihren Verwaltungskomponenten gebündelt und auf Endsystemen ausgebracht. Die Verwaltungskomponenten verwenden einfache Schnittstellen für das Management. Protokoll-spezifische Verwaltungskomponenten, die zum selben DsN gehören, werden in einer Gruppe zusammengefasst und von der entsprechenden Netz-spezifischen Verwaltungskomponente verwaltet. Im Gegensatz zu den vorgestellten Ansätzen erlaubt das Hierarchische Knoten-Management nicht nur den Austausch von Informationen, sondern erlaubt auch eine Ressourcen-Verwaltung und -Anpassung auf dem Endsystem. Darüber hinaus wird eine Hierarchie verwendet, die eine Verwaltung einer Vielzahl von Dienst-spezifischen Protokollen in heterogenen DsNs erlaubt, die zur Laufzeit hinzugefügt und entfernt werden können.

4.3 Hierarchisches Knoten-Management

In diesem Abschnitt wird das Hierarchische Knoten-Management (HKM) vorgestellt, sowie ein Überblick über die Verwaltungskomponenten und die Hierarchie gegeben. Das HKM ist ein Verwaltungssystem für DsNs auf einem Endsystem. Es erlaubt, die auf

einem Endsystem verwendeten DsNs und Dienst-spezifischen Protokolle zur Laufzeit an Netz-Eigenschaften anzupassen. Außerdem ermöglicht es, den Ressourcen-Verbrauch der auf dem Endsystem verwendeten DsNs und Dienst-spezifischen Protokolle zur Laufzeit zu regulieren. Das HKM bietet die Flexibilität, eine stets wechselnde Menge von DsNs und Protokollen auf dem Endsystem zu verwalten. Netz-spezifisches und Protokoll-spezifisches Wissen wird in Verwaltungskomponenten umgesetzt, die zur Laufzeit geladen und entfernt werden können, ohne das gesamte Verwaltungssystem anpassen zu müssen. Außerdem kann der Nutzer das Verwaltungssystem mit Richtlinien beeinflussen.

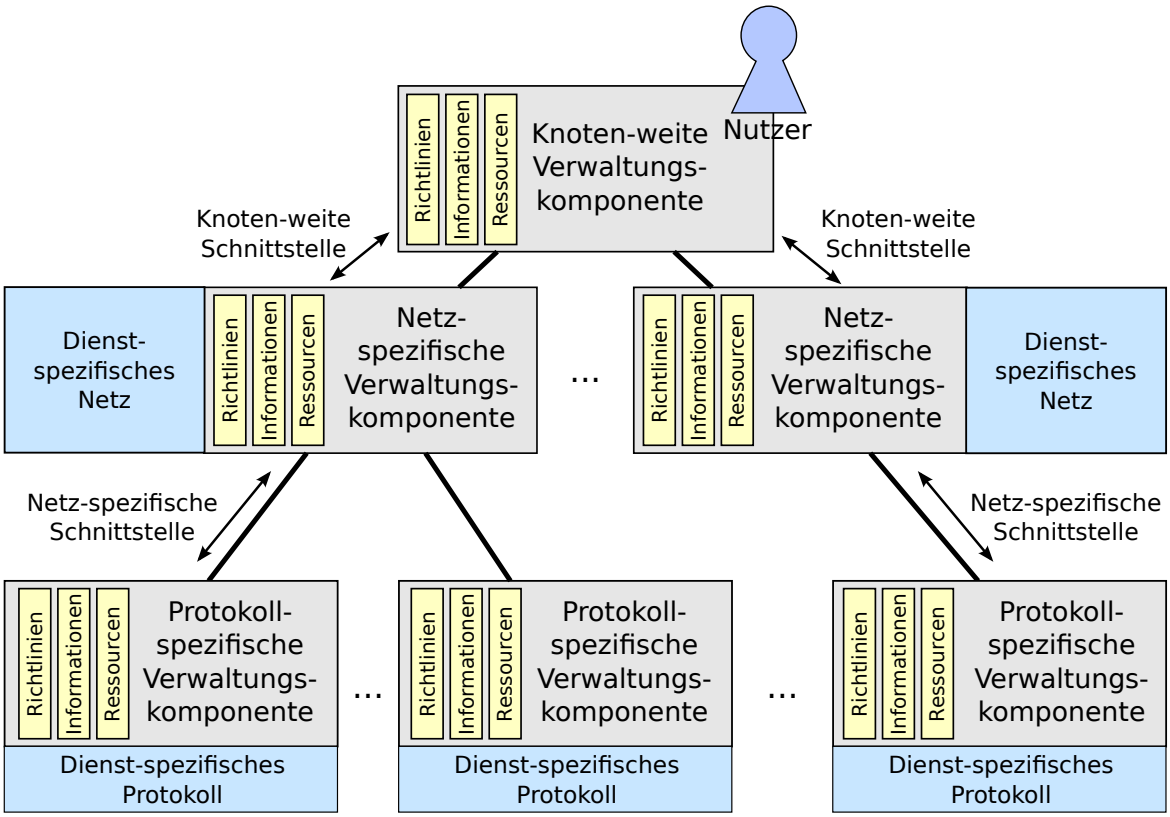

Abbildung 4.2: *Übersicht über das Hierarchische Knoten-Management*

In Abbildung 4.2 wird eine Übersicht des HKM auf einem Endsystem dargestellt. Die Verwaltungskomponenten, aus denen das Verwaltungssystem besteht, werden dargestellt. Das Verwaltungssystem besteht aus *Protokoll-spezifischen Verwaltungskomponenten*, *Netz-spezifischen Verwaltungskomponenten* und einer *Knoten-weiten Verwaltungskomponente*. Außerdem wird für jede Verwaltungskomponente gezeigt, welche Komponente sie verwaltet. Die Verwaltungskomponenten sind für *Dienst-spezifische Netze* und *Dienst-spezifische Protokolle* zuständig. Die Zuordnung der Verwaltungskomponenten zueinander wird durch die Verbindungen zwischen ihnen dargestellt. Die Interaktion zwischen den Verwaltungskomponenten wird durch Pfeile und die entsprechenden Schnittstellen,

die *Knoten-weite Schnittstelle* und *Netz-spezifische Schnittstelle*, illustriert. Darüber hinaus wird der Nutzer dargestellt, der auf das HKM über die Knoten-weite Verwaltungskomponente zugreift. Im Folgenden werden die Verwaltungskomponenten und die Interaktion zwischen den Verwaltungskomponenten entlang der Hierarchie näher vorgestellt.

4.3.1 Verwaltungskomponenten

Um Flexibilität zu erreichen, ist das HKM modular aufgebaut. Hierfür ist es in verschiedene Verwaltungskomponenten unterteilt, die verschiedene Aufgaben im Verwaltungssystem erfüllen. Diese Verwaltungskomponenten sind die Komponenten für die Protokoll-spezifische Verwaltung, die Komponenten für die Netz-spezifische Verwaltung und die Komponente für die Knoten-weite Verwaltung.

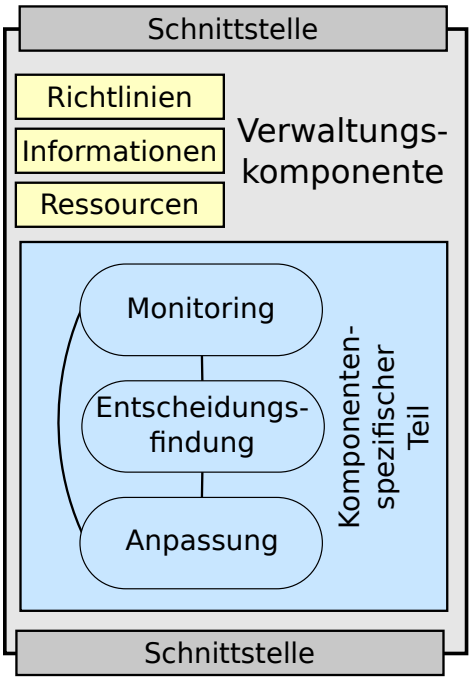

Abbildung 4.3: *Verwaltungskomponente des Hierarchischen Knoten-Managements*

In Abbildung 4.3 wird der Aufbau einer Verwaltungskomponente schematisch dargestellt. Eine Verwaltungskomponente besteht aus einer *Schnittstelle* zu einer Verwaltungskomponente auf der höheren Ebene der Hierarchie, einer *Schnittstelle* zu Verwaltungskomponenten auf der niedrigeren Ebene der Hierarchie, *Richtlinien*, *Informationen*, *Ressourcen* und einem *Komponenten-spezifischen Teil*. Der Komponenten-spezifische Teil realisiert den Management-Zyklus bestehend aus *Monitoring*, *Entscheidungsfindung* und *Anpassung*.

Eine Verwaltungskomponente interagiert mit anderen Verwaltungskomponenten über die *Schnittstellen*. Außerdem speichert sie Richtlinien, Informationen und Ressourcen. Diese werden von dem Komponenten-spezifischen Teil genutzt, um die eigentlichen Management-Aufgaben der Verwaltungskomponente zu erfüllen.

Im Folgenden werden die verschiedenen Verwaltungskomponenten des HKM näher vorgestellt.

4.3.1.1 Protokoll-spezifische Verwaltungskomponenten

In Abbildung 4.4 wird eine Protokoll-spezifische Verwaltungskomponente dargestellt. Dabei wird außerdem die Interaktion mit einem Dienst-spezifischen Protokoll schematisch anhand des gestrichelten Pfeils dargestellt. Der Komponenten-spezifische Teil einer Protokoll-spezifischen Verwaltungskomponente ist an das Dienst-spezifische Protokoll angepasst, mit dem die Verwaltungskomponente interagiert (*An Dienst-spezifisches Protokoll angepasster Teil*).

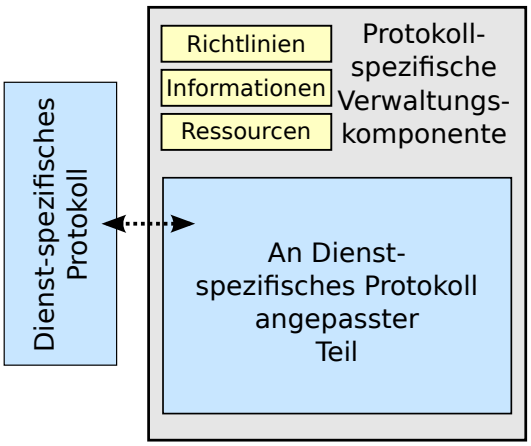

Abbildung 4.4: *Protokoll-spezifische Verwaltungskomponente des Hierarchische Knoten-Management*

Die Komponenten für die Protokoll-spezifische Verwaltung sind für die Verwaltung einzelner Dienst-spezifischer Protokolle verantwortlich. Für jede dieser Verwaltungskomponenten ist die Koppelung mit einem Dienst-spezifischem Protokoll vorgesehen. Ein Dienst-spezifisches Protokoll, das von dem HKM verwaltet wird, besitzt somit auch eine eigene Verwaltungskomponente. Diese wird für das entsprechende Dienst-spezifische Protokoll entworfen. Dadurch besitzt die Verwaltungskomponente detailliertes Wissen über das Dienst-spezifische Protokoll. Beispielsweise kennt die Verwaltungskomponente die Eigenschaften und Parameter des Dienst-spezifischen Protokolls. Außerdem weiß die Verwaltungskomponente, welche Informationen beim Monitoring des Dienst-spezifischen Protokoll erfasst werden können und welche für das Dienst-spezifische Protokoll und

dessen Betrieb tatsächlich relevant sind. Können Parameter des Protokolls an die Netz-Eigenschaften angepasst werden, werden die entsprechenden Mechanismen in der Verwaltungskomponente umgesetzt. Darüber hinaus kann die Verwaltungskomponente Wissen darüber besitzen, welche Ressourcen das Dienst-spezifische Protokoll für seinen Betrieb benötigt und wie sich der Ressourcen-Bedarf z.B. bei der Anpassung an Netz-Eigenschaften ändert. Aufgrund der engen Kopplung mit Dienst-spezifischen Protokollen ist vorgesehen, dass die Verwaltungskomponenten zusammen mit den entsprechenden Dienst-spezifischen Protokollen gebündelt und auf Endsystemen ausgebracht werden.

4.3.1.2 Netz-spezifische Verwaltungskomponenten

In Abbildung 4.5 wird eine Netz-spezifische Verwaltungskomponente dargestellt. Dabei wird außerdem die Interaktion mit einem DsN schematisch anhand des gestrichelten Pfeils dargestellt. Das DsN beinhaltet *Protokoll-spezifische Verwaltungskomponenten* und *Virtuelle Verbindungen*. Der Komponenten-spezifische Teil einer Netz-spezifischen Verwaltungskomponente ist an das Dienst-spezifische Netz angepasst, mit dem die Verwaltungskomponente interagiert (*An Dienst-spezifisches Netz angepasster Teil*).

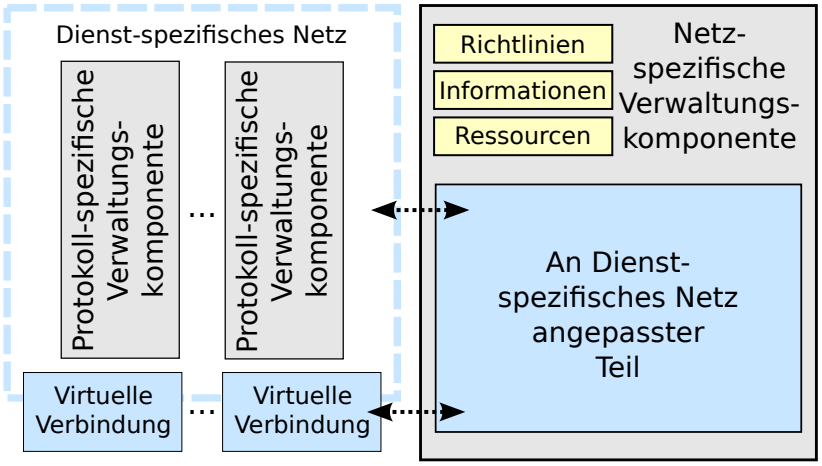

Abbildung 4.5: *Netz-spezifische Verwaltungskomponente des Hierarchische Knoten-Management*

Die Komponenten für die Netz-spezifische Verwaltung sind für die Verwaltung bestimmter DsNs verantwortlich. Für jede dieser Verwaltungskomponenten ist die Kopplung mit einem DsN bzw. dessen Komponenten auf einem Endsystem vorgesehen. Für jedes DsN, das vom HKM verwaltet wird, existiert somit auch eine eigene Verwaltungskomponente. Diese wird für das entsprechende DsN entworfen. Dadurch besitzt die Verwaltungskomponente detailliertes Wissen über das DsN. Beispielsweise weiß die Verwaltungskomponente, welche Dienst-spezifischen Protokolle in dem entsprechenden DsN eingesetzt werden. Außerdem besitzt die Verwaltungskomponente Wissen über die Virtuellen Verbindungen

und dessen Eigenschaften. Besitzen die in einem DsN eingesetzten Dienst-spezifischen Protokolle Gemeinsamkeiten oder Abhängigkeiten, ist dies der Verwaltungskomponente bekannt. Können die Eigenschaften und der Ressourcen-Bedarf der Dienst-spezifischen Protokolle eines DsN angepasst werden, können entsprechende Mechanismen in der Verwaltungskomponente umgesetzt werden. Muss beispielsweise der Ressourcen-Bedarf eines DsN auf dem Endsystem reduziert werden, weiß dadurch die Verwaltungskomponente, welche Dienst-spezifischen Protokolle angepasst werden können, um dies zu erreichen. Die Verwaltungskomponente erlaubt also eine übergreifende Verwaltung der Dienst-spezifischen Protokolle eines DsN. Aufgrund der Koppelung mit den Dienst-spezifischen Protokollen und Virtuellen Verbindungen eines DsN ist vorgesehen, dass die Verwaltungskomponenten zusammen mit den Dienst-spezifischen Protokoll-Stapeln der entsprechenden DsNs gebündelt und auf Endsystemen ausgebracht werden.

4.3.1.3 Knoten-weite Verwaltungskomponente

In Abbildung 4.6 wird die Knoten-weite Verwaltungskomponente dargestellt. Dabei wird außerdem die Interaktion mit dem Endsystem schematisch anhand des gestrichelten Pfeils und mit dem *Nutzer* über die *Nutzer-Schnittstelle* dargestellt. Das Endsystem beinhaltet die *Netz-spezifischen Verwaltungskomponenten* der DsNs, auf die das Endsystem zugreift. Der Komponenten-spezifische Teil der Knoten-weiten Verwaltungskomponente ist an das Endsystem angepasst und enthält von DsNs unabhängige Management-Funktionalität (*Knoten-weite Verwaltung*).

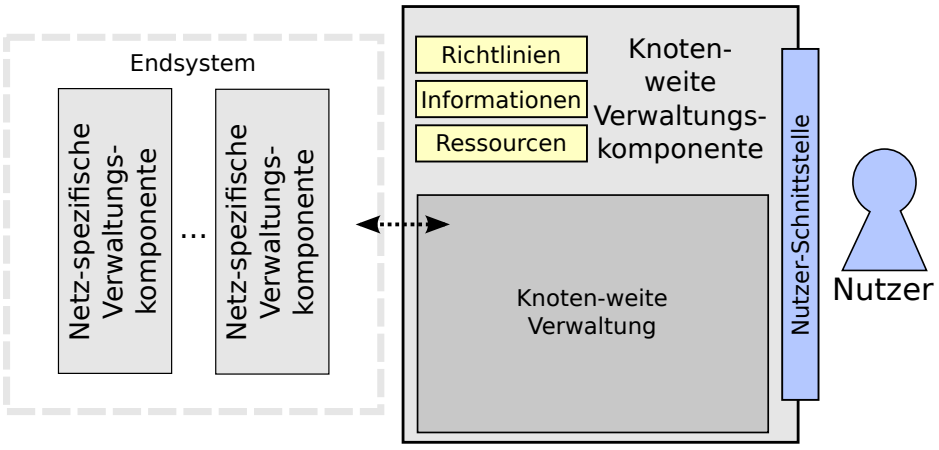

Abbildung 4.6: *Knoten-weite Verwaltungskomponente des Hierarchische Knoten-Management*

Die Komponente für die Knoten-weite Verwaltung ist für die Verwaltung des Endsystems zuständig. Diese Verwaltungskomponente existiert genau ein Mal im HKM auf einem Endsystem. Sie ist eine eigenständige Verwaltungskomponente auf dem End-

system. Sie stellt die zentrale Verwaltungskomponente des HKM dar und ist somit die wichtigste der Verwaltungskomponenten. Um Flexibilität zu erreichen, ist sie unabhängig von Dienst-spezifischen Protokollen und den DsNs entworfen. Das bedeutet, die Funktionsfähigkeit oder die Funktionsweise dieser Verwaltungskomponente hängt nicht von dem Vorhandensein bestimmter DsNs oder Dienst-spezifischer Protokolle auf dem Endsystem ab. Allerdings sind die anderen Verwaltungskomponenten auf die Knoten-weite Verwaltungskomponente angewiesen. Ohne sie funktioniert das Verwaltungssystem nicht. Die Knoten-weite Verwaltungskomponente sammelt Informationen, die sie von anderen Verwaltungskomponenten erhält, und kann somit grob-granulares Wissen über das Endsystem erhalten. Anpassungen in den anderen Verwaltungskomponenten können durch die Knoten-weite Verwaltungskomponente initiiert werden, z.B. durch Nutzer-Richtlinien oder Ressourcen-Zuteilungen. Außerdem ist die Verwaltungskomponente für die Ressourcen-Verwaltung hauptverantwortlich. Wird die Verwaltungskomponente auf einem Endsystem gestartet, bindet sie sämtliche verfügbaren Ressourcen an sich. Andere Verwaltungskomponenten können Ressourcen von dieser Komponente erhalten, wie in Abschnitt 4.6 beschrieben. Aufgrund der zentralen Rolle der Knoten-weiten Verwaltungskomponente ist vorgesehen, dass diese Verwaltungskomponente an einer zentralen Stelle wie z.B. in dem Rahmenwerk für DsNs (siehe Abschnitt 4.7) auf Endsystemen ausgebracht wird.

4.3.2 Hierarchie und Interaktion zwischen Verwaltungskomponenten

Die Verwaltungskomponenten sind in einer Hierarchie angeordnet und interagieren miteinander entlang dieser Hierarchie. In diesem Abschnitt wird diese Hierarchie näher beschrieben und näher auf die Interaktion zwischen den Verwaltungskomponenten mit Hilfe der Schnittstellen eingegangen.

In Abbildung 4.7 wird die Hierarchie des Verwaltungssystems erneut dargestellt. Es werden die *Protokoll-spezifischen Verwaltungskomponenten*, die *Netz-spezifische Verwaltungskomponenten* und die Knoten-weite Verwaltungskomponente gezeigt. Für die Netz-spezifischen Verwaltungskomponenten werden die *Dienst-spezifischen Netze*, für die sie verantwortlich sind, dargestellt. Für die Protokoll-spezifischen Verwaltungskomponenten werden die *Dienst-spezifischen Protokolle*, für die sie verantwortlich sind, dargestellt. Die Zuordnung der Verwaltungskomponenten zueinander und damit die Hierarchie werden mit Hilfe der Verbindungslinien zwischen den Verwaltungskomponenten dargestellt. Außerdem wird die Interaktion zwischen den Verwaltungskomponenten mit Pfeilen und den entsprechenden Schnittstellen illustriert. Die *Knoten-weite Schnittstelle* wird zwischen der Knoten-weiten Verwaltungskomponente und den Netz-spezifischen Verwaltungskomponenten verwendet. Die *Netz-spezifische Schnittstelle* wird zwischen einer Netz-spezifischen Verwaltungskomponente und den ihr zugeordneten Protokoll-spezifischen Verwaltungskomponenten verwendet.

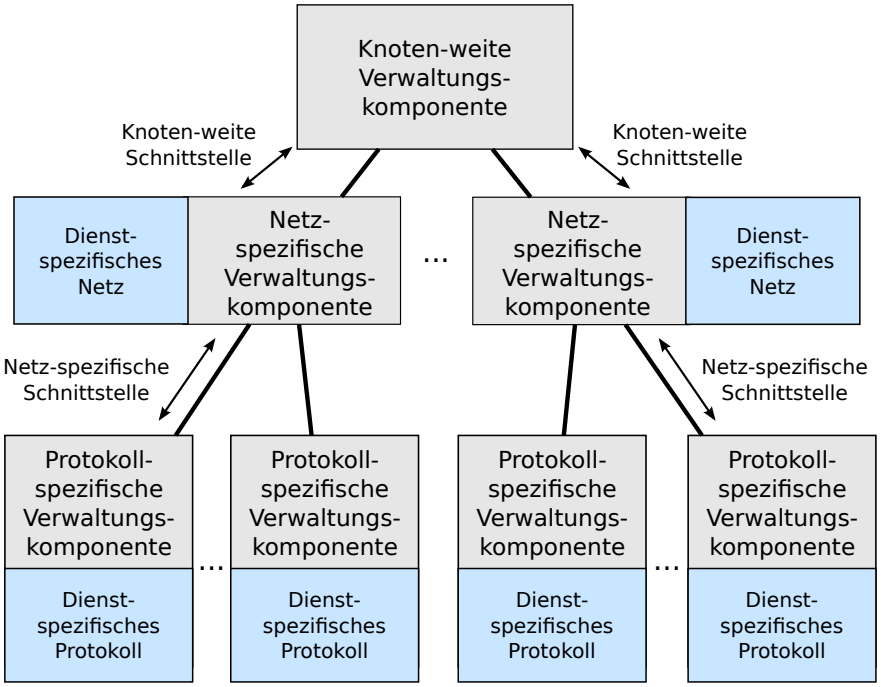

Abbildung 4.7: *Hierarchie des Verwaltungssystem und Interaktion zwischen Verwaltungskomponenten*

Die Struktur, in der die Verwaltungskomponenten angeordnet sind, entspricht einem Baum. An der obersten Ebene der Hierarchie befindet sich die Knoten-weite Verwaltungskomponente. Jede Netz-spezifische Verwaltungskomponente ist mit der Knoten-weiten Verwaltungskomponente verbunden. Eine direkte Verbindung zwischen Netz-spezifischen Verwaltungskomponenten ist nicht vorgesehen. Dadurch benötigt eine Netz-spezifische Verwaltungskomponente kein Wissen über weitere Netz-spezifische Verwaltungskomponenten auf dem Endsystem. Jede Protokoll-spezifische Verwaltungskomponente ist mit genau einer Netz-spezifischen Verwaltungskomponente verbunden. Eine direkte Verbindung zwischen Protokoll-spezifischen Verwaltungskomponenten ist nicht vorgesehen. Dadurch benötigt eine Protokoll-spezifische Verwaltungskomponente kein Wissen über weitere Protokoll-spezifische Verwaltungskomponenten auf dem Endsystem. Die Protokoll-spezifischen Verwaltungskomponenten sind außerdem nicht mit beliebigen Netz-spezifischen Verwaltungskomponenten verbunden. Die Verbindungen orientieren sich an den DsNs und Dienst-spezifischen Protokollen. Eine Protokoll-spezifische Verwaltungskomponente ist nur dann mit einer Netz-spezifischen Verwaltungskomponente verbunden, wenn das entsprechende Dienst-spezifische Protokoll auch in dem entsprechenden DsN verwendet wird.

4.3.2.1 Knoten-weite Schnittstelle

Die Verwaltungskomponenten interagieren miteinander, z.B. um Informationen auszutauschen, Ressourcen anzufordern oder wieder freizugeben. Die Interaktion der Verwaltungskomponenten verläuft entlang der zuvor beschriebenen Hierarchie.

Für die Interaktion verwenden die Verwaltungskomponenten gemeinsame Schnittstellen. Die Schnittstelle, die zwischen der Knoten-weiten Verwaltungskomponente und den Netz-spezifischen Verwaltungskomponenten verwendet wird besteht aus einfachen Funktionen. Diese Funktionen werden im Folgenden aufgelistet.

- *request(resources)*
- *grant(resources)*
- *withdraw(resources)*
- *inform(information)*
- *adapt(task)*
- *register()*
- *unregister()*

Die Funktion *request* wird verwendet, um Ressourcen von einer anderen Verwaltungskomponente anzufordern. Die angeforderten Ressourcen werden durch den Parameter *resources* angegeben. Die Funktion *grant* wird verwendet, um Ressourcen einer anderen Verwaltungskomponente zuzuteilen. Die zugeteilten Ressourcen werden durch den Parameter *resources* angegeben. Die Funktion *withdraw* wird verwendet, um einer anderen Verwaltungskomponente Ressourcen zu entziehen. Die entzogenen Ressourcen werden durch den Parameter *resources* angegeben. Die Funktion *inform* wird verwendet, um mit einer anderen Verwaltungskomponente Informationen auszutauschen. Die dabei übermittelten Informationen werden durch den Parameter *information* angegeben. Die Funktion *adapt* wird verwendet, um Anpassungen in einer anderen Verwaltungskomponente anzufordern. Die angeforderten Anpassungen werden durch den Parameter *task* angegeben. Die Funktion *register* wird von einer Verwaltungskomponente verwendet, um sich bei einer anderen Verwaltungskomponente zu registrieren. Die Funktion *unregister* wird von einer bereits registrierten Verwaltungskomponente verwendet, um sich von einer Verwaltungskomponente abzumelden.

Bei der Interaktion zwischen den Verwaltungskomponenten über diese Schnittstelle, ist die Verwendung der verschiedenen Funktionen nicht in jeder Verwaltungskomponente vorgesehen. Netz-spezifische Verwaltungskomponenten verwenden bei der Interaktion mit der Knoten-weiten Verwaltungskomponente die Funktionen *request* und *inform*.

Über *request* können sie Ressourcen von der Knoten-weiten Verwaltungskomponente anfordern. Über *inform* können sie Informationen an die Knoten-weite Verwaltungskomponente übergeben. Die Knoten-weite Verwaltungskomponente verwendet bei der Interaktion mit Netz-spezifischen Verwaltungskomponenten die Funktionen *grant*, *withdraw* und *adapt*. Über *grant* kann sie Netz-spezifischen Verwaltungskomponenten Ressourcen zuteilen. Über *withdraw* kann sie den Netz-spezifischen Verwaltungskomponenten zuvor zugewiesene Ressourcen wieder entziehen. Über *adapt* kann sie Anpassungen in Netz-spezifischen Verwaltungskomponenten anstoßen.

Alle Netz-spezifischen Verwaltungskomponenten müssen diese einheitliche Schnittstelle verwenden, damit sie im HKM genutzt werden können. Darüber hinaus müssen die in den Parametern übergebenen Ressourcen, Informationen und Aufgaben so formatiert sein, dass sie von den entsprechenden Verwaltungskomponenten interpretiert werden können.

4.3.2.2 Netz-spezifische Schnittstellen

Die Schnittstelle, die für die Interaktion zwischen den Netz-spezifischen Verwaltungskomponenten und den Protokoll-spezifischen Verwaltungskomponenten verwendet wird, besteht aus den selben Funktionen wie die Schnittstelle für die Interaktion zwischen der Knoten-weiten Verwaltungskomponente und den Netz-spezifischen Verwaltungskomponenten. Auch die Richtungen, in denen die Funktionen verwendet werden, ist identisch. Protokoll-spezifische Verwaltungskomponenten können Ressourcen von Netz-spezifischen Verwaltungskomponenten anfordern und Informationen an Netz-spezifische Verwaltungskomponenten weiterreichen. Eine Netz-spezifische Verwaltungskomponente kann denen ihr zugeordneten Protokoll-spezifischen Verwaltungskomponenten Ressourcen zuteilen und zuvor zugewiesene Ressourcen entziehen oder Anpassungen anfordern. Der Unterschied zur Schnittstelle zwischen der Knoten-weiten Verwaltungskomponente und den Netz-spezifischen Verwaltungskomponenten ist allerdings, dass diese Schnittstelle nicht einheitlich sein muss. Die Netz-spezifische Verwaltungskomponente und die Protokoll-spezifischen Verwaltungskomponenten des selben DsN werden für das entsprechende DsN bzw. die entsprechenden Dienst-spezifischen Protokolle entworfen. Dadurch ist es möglich, zwischen diesen Verwaltungskomponenten eigene, auf die Anforderungen und Eigenschaften des DsN angepasste Schnittstellen zu verwenden. Hierdurch wird die Flexibilität des HKM erhöht. Werden eigene Schnittstellen zwischen den Verwaltungskomponenten auf diesen beiden Hierarchie-Ebenen verwendet, führt die Netz-spezifische Verwaltungskomponente die Umsetzung der Formate der über die Schnittstelle mit der Knoten-weiten Verwaltungskomponente ausgetauschten Ressourcen, Informationen und Aufgaben durch.

4.4 Überwachung Dienst-spezifischer Netze

Der Zustand eines Endsystems kann sich über dessen Laufzeit verändern. Beispielsweise können sich die Anzahl der Anwendungen, die Anzahl der DsNs, auf die das Endsystem zugreift, der Ressourcen-Verbrauch der verschiedenen Komponenten auf dem Endsystem, oder die Eigenschaften der Netz-Infrastruktur, über die das Endsystem kommuniziert, verändern. Damit das Endsystem auf solche Veränderungen auf geeignete Weise reagieren kann, z.B. durch Anpassungen von Protokoll-Parametern an die Netz-Eigenschaften, werden Informationen über den Zustand des Endsystems benötigt. Um diese Informationen zu erhalten, muss das Endsystem die verwendeten Dienst-spezifischen Protokoll-Stapel bzw. Dienst-spezifischen Protokolle, die Virtuellen Verbindungen und die Anwendungen zur Laufzeit überwachen. Hierfür überwacht jede Verwaltungskomponente im HKM den aktuellen Zustand zur Laufzeit (so genanntes Monitoring). Die dabei erhaltenen Informationen werden entlang der Hierarchie mit anderen Verwaltungskomponenten ausgetauscht. Die von anderen Verwaltungskomponenten erhaltenen Informationen werden mit den eigenen zusammengefasst (so genannte Aggregation) und wiederum weitergegeben. Dadurch erhält das HKM flexibel und effizient Informationen über den Zustand des Endsystems.

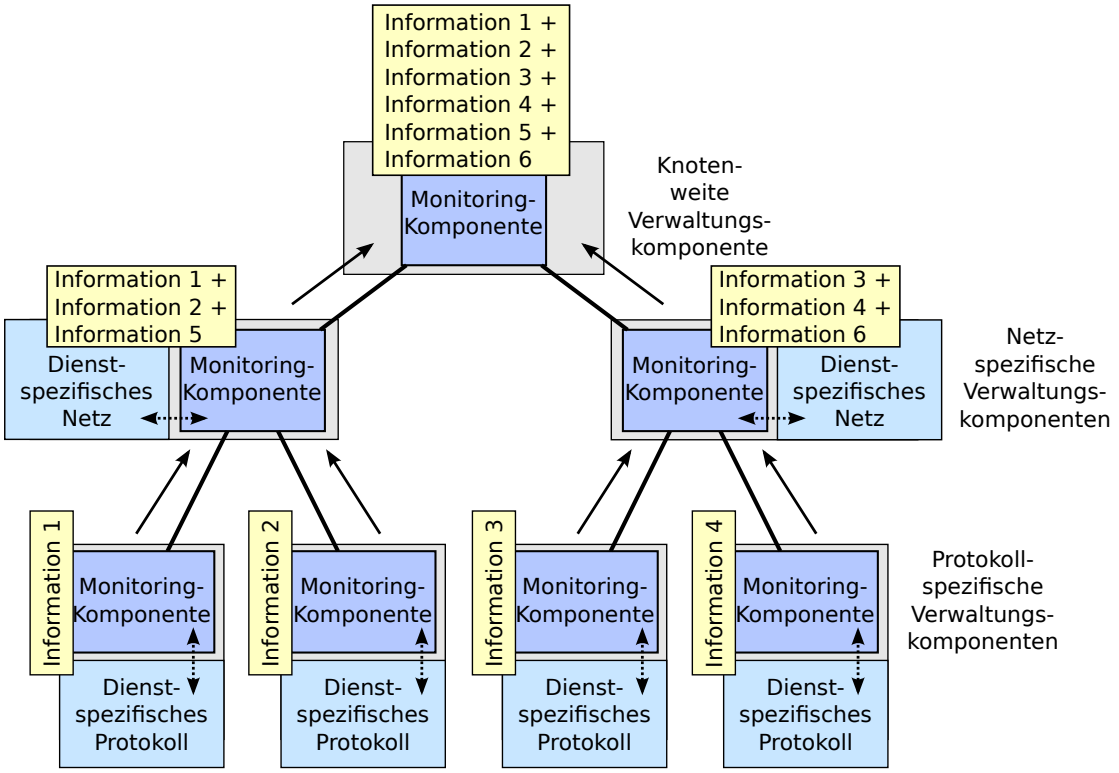

Abbildung 4.8: *Monitoring und Aggregation im Hierarchischen Knoten-Management.*

In Abbildung 4.8 wird schematisch dargestellt, wie das Monitoring und die Aggregation von Informationen entlang der Hierarchie funktioniert. Es werden die Verwaltungskomponenten der verschiedenen Hierarchie-Ebenen und deren Verbindungen miteinander gezeigt. Jede Verwaltungskomponente enthält eine Monitoring-Komponente im Komponenten-spezifischen Teil. Die Monitoring-Komponente in einer Protokoll-spezifischen Verwaltungskomponente überwacht das entsprechende Dienst-spezifische Protokoll (dargestellt durch die vertikalen gestrichelten Pfeile). Die Monitoring-Komponente in einer Netz-spezifischen Verwaltungskomponente überwacht die Virtuelle Verbindung und die Protokoll-spezifischen Verwaltungskomponenten (dargestellt durch die horizontalen gestrichelten Pfeile). Die Monitoring-Komponente in der Knoten-weiten Verwaltungskomponente überwacht das Endsystem und die Netz-spezifischen Verwaltungskomponenten. Entlang der Hierarchie tauschen die Monitoring-Komponenten die ihnen zur Verfügung stehenden Informationen (repräsentiert durch die gelben Kästen) mit Hilfe der Schnittstellen-Funktion *inform()* aus, was durch die durchgezogenen Pfeile dargestellt wird. Die in einer Monitoring-Komponente erhaltenen Informationen werden aggregiert, was exemplarisch durch die Addition der Informationen in den gelben Kästen dargestellt wird.

Jede Verwaltungskomponente führt ein eigenständiges Monitoring durch, das spezifisch für die entsprechende Verwaltungskomponente ist. Verwaltungskomponenten auf niedrigeren Ebenen der Hierarchie senden ihre gesammelten Informationen an die Verwaltungskomponenten, die sich auf der nächst höheren Ebene der Hierarchie befinden. Erhält eine Verwaltungskomponente Informationen einer anderen Verwaltungskomponente, aggregiert sie diese mit den gesammelten Informationen. Die aggregierten Informationen reicht die Verwaltungskomponente wiederum weiter an die Verwaltungskomponente auf der höheren Hierarchie-Ebene.

Protokoll-spezifische Verwaltungskomponente: Die Monitoring-Komponente in einer Protokoll-spezifischen Verwaltungskomponente überwacht das Dienst-spezifische Protokoll, für das die Verwaltungskomponente zuständig ist. Dabei kann sie beispielsweise Informationen über die verschiedenen Verbindungen bzw. Datenströme, die über das Dienst-spezifische Protokoll betrieben werden, und den Ressourcen-Verbrauch des Dienst-spezifischen Protokolls überwachen. Diese Informationen fasst die Monitoring-Komponente zusammen und reicht sie über die Funktion *inform()* an die Netz-spezifische Verwaltungskomponente, die für die Protokoll-spezifischen Verwaltungskomponente zuständig ist, weiter.

Netz-spezifische Verwaltungskomponente: Die Monitoring-Komponente in einer Netz-spezifischen Verwaltungskomponente überwacht die Virtuellen Verbindungen des DsN, für das die Verwaltungskomponente zuständig ist. Dabei kann sie beispielsweise den aktuellen Ressourcen-Verbrauch der Virtuellen Verbindung ermitteln. Außerdem überwacht die Verwaltungskomponente die Protokoll-spezifischen Verwaltungskomponenten, für die sie zuständig ist. Die Monitoring-Komponente sammelt hierfür die Informatio-

nen, die sie von den Protokoll-spezifischen Verwaltungskomponenten erhält. Dadurch erhält sie beispielsweise eine aggregierte Sicht auf die aktiven Datenströme und auf den Ressourcen-Verbrauch in den Dienst-spezifischen Protokollen. Die gesammelten Informationen aggregiert die Monitoring-Komponente und reicht sie über die Funktion *inform()* an die Knoten-weite Verwaltungskomponente weiter.

Knoten-weite Verwaltungskomponente: Die Monitoring-Komponente in der Knoten-weiten Verwaltungskomponente überwacht den Zustand des Endsystems sowie die auf dem Endsystem verwendeten DsNs und Anwendungen. Die Monitoring-Komponente überwacht beispielsweise, welche Anwendungen verwendet werden und ob sich die dem Endsystem zur Verfügung stehenden Ressourcen ändern. Werden z.B. neue Netzwerkschnittstellen verfügbar oder ändern sich die maximalen Datenraten bereits vorhandener Netzwerkschnittstellen, wird dies von der Monitoring-Komponente erkannt. Außerdem überwacht die Monitoring-Komponente die Netz-spezifischen Verwaltungskomponenten, die auf dem Endsystem verwendet werden. Die Monitoring-Komponente sammelt hierfür die Informationen, die sie von den Netz-spezifischen Verwaltungskomponenten erhält. Dadurch erhält sie beispielsweise eine aggregierte Sicht auf die aktiven Datenströme und auf den Ressourcen-Verbrauch in den DsNs.

4.5 Nutzer-Interaktion und -Richtlinien

Nutzer können Informationen, z.B. über die verwendeten DsNs und Protokolle, den Ressourcen-Verbrauch oder die aktiven Datenströme aus dem HKM abrufen. Hierfür rufen sie die durch das Monitoring erhaltenen Informationen ab. Außerdem erlaubt das HKM dem Nutzer, die Verwaltung des Endsystems zu beeinflussen. Hierfür kann der Nutzer Richtlinien festlegen, die bei der Verwaltung genutzt werden. Durch die Richtlinien kann der Nutzer beispielsweise festlegen, welche Datenströme oder DsNs ihm wichtiger sind. Dies ist insbesondere für die später beschriebene Ressourcen-Verwaltung interessant, da der Nutzer dadurch Prioritäten bei der Verteilung der Ressourcen des Endsystems an die verschiedenen Komponenten festlegen kann.

In Abbildung 4.9 wird die Interaktion des Nutzers mit dem HKM und die Verwendung von Nutzer-Richtlinien schematisch dargestellt. Der Nutzer interagiert mit dem HKM über die Knoten-weite Verwaltungskomponente. Hierzu verwendet er die Nutzer-Schnittstelle. Der Nutzer ruft über diese Schnittstelle Informationen ab. Außerdem übergibt er über diese Schnittstelle Richtlinien an das HKM. Das HKM verteilt diese Richtlinien entlang der Hierarchie an die Verwaltungskomponenten, was durch die Pfeile illustriert wird. Dabei werden die Richtlinien in den Verwaltungskomponenten verfeinert. Die in den Verwaltungskomponenten vorliegenden Richtlinien werden durch gelbe Kästen dargestellt.

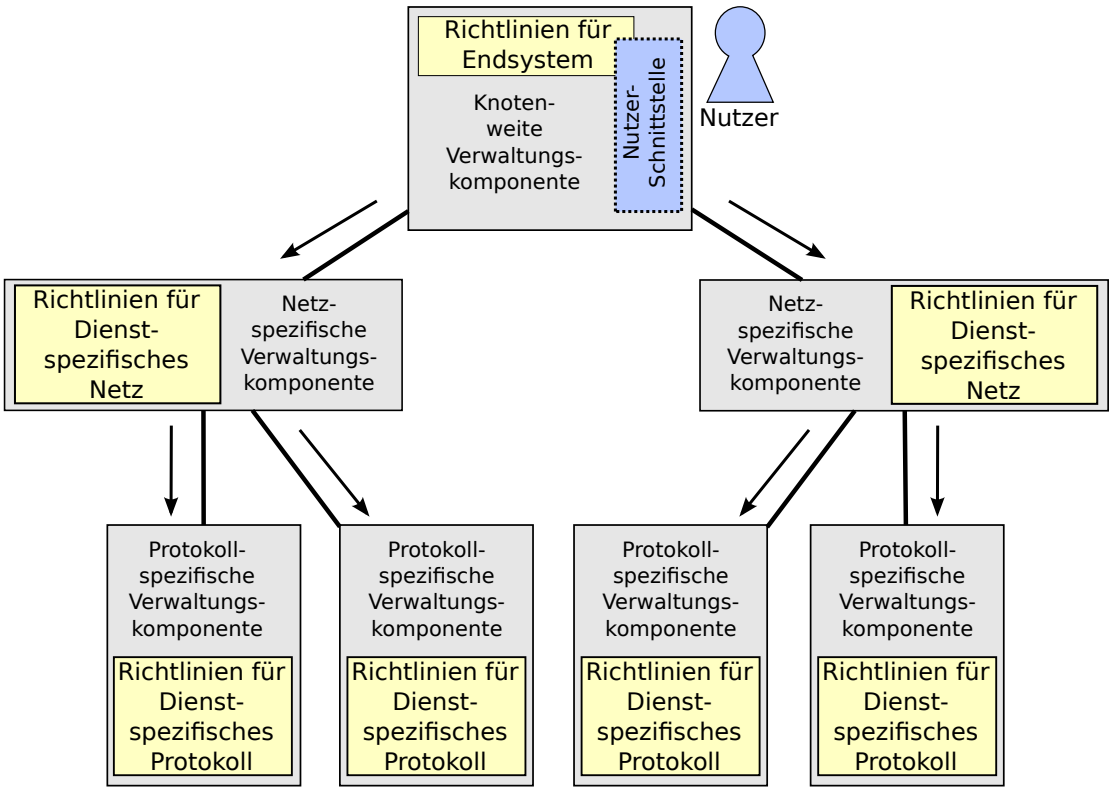

Abbildung 4.9: *Nutzer-Interaktion und -Richtlinien im Hierarchischen Knoten-Management.*

Ruft der Nutzer Informationen ab, bezieht die Knoten-weite Verwaltungskomponente die entsprechenden Informationen aus ihrer Monitoring-Komponente. Der Nutzer kann also die Informationen abrufen, die auch der Knoten-weiten Verwaltungskomponente vorliegen. Der Nutzer erhält somit eine aggregierte Sicht auf den Zustand des Endsystems.

Der Nutzer kann Richtlinien definieren, die das Verhalten des HKM beeinflussen. Ein Beispiel für eine solche Nutzer-Richtlinie ist die Vergabe von Prioritäten an die Datenströme bestimmter Anwendungen oder an bestimmte DsNs. Hierdurch werden in der später beschriebenen Ressourcen-Verwaltung die entsprechenden Datenströme oder DsNs bevorzugt bei der Ressourcen-Zuteilung behandelt. Der Nutzer übergibt seine Richtlinie über die Nutzer-Schnittstelle an das HKM. Die Richtlinie wird in den einzelnen Verwaltungskomponenten verarbeitet und entlang der Hierarchie weitergereicht. Dabei werden die Richtlinien entlang der Hierarchie schrittweise verfeinert. Die Knoten-weite Verwaltungskomponente nimmt die Richtlinie vom Nutzer entgegen und verarbeitet sie. Die Knoten-weite Verwaltungskomponente gibt die Richtlinie an die davon betroffenen Netz-spezifischen Verwaltungskomponenten weiter. Dabei werden an eine Netz-spezifische Verwaltungskomponente auch nur die für die entsprechende Verwaltungskomponente relevanten Teile der Nutzer-Richtlinie weitergegeben. Die Netz-

spezifische Verwaltungskomponente verarbeitet die Nutzer-Richtlinie weiter. Außerdem übergibt sie die Nutzer-Richtlinie an alle davon betroffenen Protokoll-spezifischen Verwaltungskomponenten weiter. Dabei werden an eine Protokoll-spezifische Verwaltungskomponente nur die für die entsprechende Verwaltungskomponente relevanten Teile der Nutzer-Richtlinie weitergegeben. Die Protokoll-spezifische Verwaltungskomponente verarbeitet die Nutzer-Richtlinie schließlich weiter.

Für die Bestimmung der von einer Richtlinie betroffenen Verwaltungskomponenten können die Verwaltungskomponenten die über das Monitoring gesammelten Informationen nutzen. Betrifft eine Richtlinie beispielsweise einen bestimmten Datenstrom, werden die durch das Monitoring gesammelten Informationen folgendermaßen benutzt: Die Knoten-weite Verwaltungskomponente besitzt durch ihre aggregierte Sicht auf das Endsystem die Information, zu welchem DsN der Datenstrom gehört. Dadurch kann die Richtlinie an die entsprechende Netz-spezifische Verwaltungskomponente weitergegeben werden. Die Netz-spezifische Komponente besitzt durch das Monitoring die Information, zu welchem Dienst-spezifischen Protokoll der Datenstrom gehört. Dadurch kann die Richtlinie schließlich an die entsprechende Protokoll-spezifische Verwaltungskomponente weitergegeben werden.

4.6 Ressourcen-Verwaltung

Das HKM erlaubt eine flexible Verwaltung der Ressourcen eines Endsystems. Die Ressourcen-Verwaltung nutzt die in den vorherigen Abschnitten beschriebenen Verwaltungskomponenten und die Hierarchie. Wie die Ressourcen-Verwaltung funktioniert, wird in diesem Abschnitt näher beschrieben.

Die Ressourcen-Verwaltung wird in Abbildung 4.10 schematisch dargestellt. Es wird die Management-Hierarchie mit ihren Verwaltungskomponenten und verwalteten DsNs und Dienst-spezifischen Protokollen gezeigt. In jeder Verwaltungskomponente sind die ihr zur Verfügung stehenden Ressourcen abgebildet. Die Zuteilung von Ressourcen entlang der Hierarchie wird mit Hilfe der durchgezogenen Pfeile und der verwendeten Funktionen dargestellt. Die Interaktion von Verwaltungskomponenten mit DsNs und Dienst-spezifischen Protokollen für die Umsetzung der Ressourcen-Zuteilungen wird mit gestrichelten Pfeilen dargestellt.

Die Verwaltungskomponenten verwalten die ihnen zur Verfügung stehenden Ressourcen. Die Knoten-weite Verwaltungskomponente verwaltet die Ressourcen des Endsystems (*Ressourcen für Endsystem*), jede Netz-spezifische Verwaltungskomponente verwaltet die Ressourcen des entsprechenden DsN (*Ressourcen für Dienst-spezifisches Netz*). Jede Protokoll-spezifische Verwaltungskomponente verwaltet die Ressourcen des entsprechenden Dienst-spezifischen Protokolls (*Ressourcen für Dienst-spezifisches Protokoll*). Die Verwaltungskomponenten interagieren entlang der Hierarchie, um sich gegenseitig

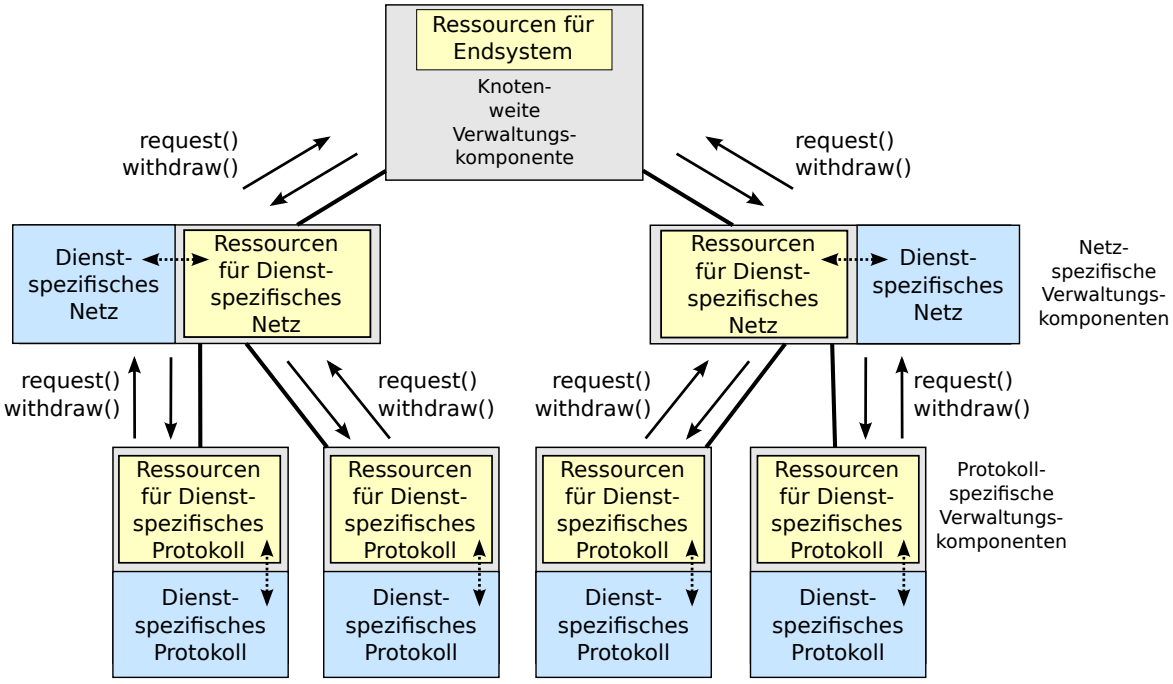

Abbildung 4.10: *Zuweisung und Entziehen von Ressourcen im Hierarchischen Knoten-Management*

Ressourcen zuzuteilen oder zu entziehen. Verwaltungskomponenten auf niedrigeren Schichten der Hierarchie fordern Ressourcen von Verwaltungskomponenten höherer Schichten an (*request()*). Verwaltungskomponenten auf höheren Schichten der Hierarchie vergeben Ressourcen an Verwaltungskomponenten niedrigerer Schichten. Außerdem entziehen Verwaltungskomponenten höherer Schichten bei Bedarf den Verwaltungskomponenten niedrigerer Schichten zuvor zugewiesene Ressourcen wieder (*withdraw()*). Die Verwaltungskomponenten parametrisieren die ihnen zugeordneten DsNs und Protokolle so, dass sie die ihnen zugeteilten Ressourcen nutzen bzw. Beschränkungen nicht überschreiten.

Interaktion

Für die Zuteilung von Ressourcen an Verwaltungskomponenten wird die in Abschnitt 4.3.2 beschriebene Schnittstelle zwischen den entsprechenden Verwaltungskomponenten verwendet. Eine Verwaltungskomponente fordert mit Hilfe der Funktion *request()* Ressourcen von der Verwaltungskomponente auf der höheren Schicht der Hierarchie an. Die Verwaltungskomponente auf der höheren Schicht der Hierarchie entscheidet darüber, ob und wie viele Ressourcen genau der Verwaltungskomponente auf der niedrigeren Schicht zugeteilt werden, und weist sie der Verwaltungskomponente über die Schnittstelle zu. Benötigt eine Verwaltungskomponente bereits zugewiesene Ressourcen selbst, kann sie

diese über die Schnittstelle mit Hilfe der Funktion *withdraw()* wieder zurückfordern. Die Verwaltungskomponente, der die Ressourcen zuvor zugewiesen wurden, gibt diese daraufhin wieder frei.

Entscheidungsfindung

Die Entscheidung, ob und wie viele Ressourcen die Verwaltungskomponente einer anderen Verwaltungskomponente zuweist, ist Aufgabe der Verwaltungskomponente selbst. Ebenso obliegt die Entscheidung, ob und wieviele Ressourcen eine Verwaltungskomponente von einer anderen anfordert, der Verwaltungskomponente selbst. Daher werden die Mechanismen für diese Entscheidungen in jeder Verwaltungskomponente selbst realisiert.

4.6.1 Initialisierung

Die Initialisierung der Ressourcen-Verwaltung geschieht zusammen mit der Initialisierung der Knoten-weiten Verwaltungskomponente. Die Knoten-weite Verwaltungskomponente reserviert alle verfügbaren Ressourcen des Endsystems. Dabei speichert die Knoten-weite Verwaltungskomponente die verfügbaren Prozessor-, Speicher- und Netz-Ressourcen ab. Daraufhin ist die Ressourcen-Verwaltung bereit und die Knoten-weite Verwaltungskomponente wartet auf Ressourcen-Anfragen von Netz-spezifischen Verwaltungskomponenten.

4.6.2 Hinzufügen Dienst-spezifischer Netze

Wird eine Verwaltungskomponente zu dem HKM hinzugefügt, registriert sich diese Verwaltungskomponente bei der Verwaltungskomponente, die sich über ihr in der Hierarchie befindet. Dadurch weiß die übergeordnete Verwaltungskomponente, dass die untergeordnete existiert. Außerdem kann die untergeordnete Verwaltungskomponente der übergeordneten dabei ihren Ressourcenbedarf mitteilen. Hierdurch kann die übergeordnete Verwaltungskomponente der untergeordneten direkt die benötigten Ressourcen zuteilen.

Wenn ein Endsystem auf ein DsN zugreift, baut es eine Virtuelle Verbindung zu dem DsN auf und lädt die entsprechenden Dienst-spezifischen Protokolle. Dabei wird die Netz-spezifische Verwaltungskomponente des DsN geladen. Außerdem werden die Protokoll-spezifischen Verwaltungskomponenten für die Dienst-spezifischen Protokolle in dem DsN geladen. Die Protokoll-spezifischen Verwaltungskomponenten registrieren sich bei der ihnen zugeordneten Netz-spezifischen Verwaltungskomponente. Dabei fordern sie Ressourcen von der Netz-spezifischen Verwaltungskomponente an. Dies setzt voraus, dass die Protokoll-spezifischen Verwaltungskomponenten wissen, welche Prozessor-, Speicher- und Netz-Ressourcen das entsprechende Dienst-spezifische Protokoll benötigt. Durch die Ressourcen-Anforderung der Protokoll-spezifischen Verwaltungskomponenten erhält die

Netz-spezifische Verwaltungskomponente Wissen über den Ressourcen-Bedarf für das DsN. Diesem Ressourcen-Bedarf fügt die Netz-spezifische Verwaltungskomponente selbst benötigte Ressourcen-Anforderungen hinzu. Dies setzt dementsprechend voraus, dass die Netz-spezifische Verwaltungskomponente weiß, welche Prozessor-, Speicher- und Netz-Ressourcen sie benötigt. Die Netz-spezifische Verwaltungskomponente registriert sich bei der Knoten-weiten Verwaltungskomponente. Dabei übergibt sie der Knoten-weiten Verwaltungskomponente den ermittelten Ressourcen-Bedarf.

Die Knoten-weite Verwaltungskomponente ermittelt anhand der Ressourcen des Endsystems und der Ressourcen-Zuteilungen an Netz-spezifische Verwaltungskomponenten, ob ausreichend Ressourcen auf dem Endsystem vorhanden sind, um den Ressourcen-Bedarf der Netz-spezifischen Verwaltungskomponente abzudecken. Trifft dies zu, weist die Knoten-weite Verwaltungskomponente der Netz-spezifischen Verwaltungskomponente die Ressourcen zu. Hierfür speichert sie die Ressourcen-Reservierungen und teilt der Netz-spezifischen Komponente die zugeteilten Ressourcen mit.

Die Netz-spezifische Verwaltungskomponente speichert die Ressourcen-Zuteilung. Außerdem teilt sie die Ressourcen, basierend auf den Ressourcen-Anforderungen, auf die verschiedenen Protokoll-spezifischen Verwaltungskomponenten auf. Diese Zuteilungen werden ebenfalls in der Netz-spezifischen Verwaltungskomponente gespeichert. Die Netz-spezifische Verwaltungskomponente teilt den Protokoll-spezifischen Verwaltungskomponenten die ihnen zugewiesenen Ressourcen mit.

Die Protokoll-spezifischen Verwaltungskomponenten speichern die Ressourcen-Zuteilung und verwenden schließlich die Ressourcen für den Betrieb der entsprechenden Dienst-spezifischen Protokolle.

4.6.3 Betrieb Dienst-spezifischer Netze und Ressourcen-Anpassungen

Während des Betriebs führt jede Verwaltungskomponente eine Überwachung ihres tatsächlichen Ressourcen-Verbrauchs durch. Benötigt eine Verwaltungskomponente mehr Ressourcen, als ihr zugeteilt worden sind, wird außerdem eine Anpassung der Ressourcen-Zuteilung durchgeführt.

Überwachung des Ressourcen-Verbrauchs

Solange Verwaltungskomponenten verwendet werden, wird eine Überwachung des tatsächlichen Ressourcen-Bedarfs in den Verwaltungskomponenten durchgeführt. Jede Verwaltungskomponente überwacht ihren eigenen Ressourcen-Verbrauch. Außerdem wird der eigene Ressourcen-Verbrauch an die Verwaltungskomponente auf der höheren Ebene der Hierarchie mit Hilfe der Schnittstellen-Funktion *inform()* übermittelt. Eine Verwaltungskomponente, die den Ressourcen-Verbrauch von anderen Verwaltungskomponenten erhält, führt eine Aggregation der erhaltenen Informationen durch. Der aggregierte

Ressourcen-Verbrauch wird zusätzlich mit dem eigenen Verbrauch zusammengefasst. Der gesamte Ressourcen-Verbrauch wird an die übergeordnete Verwaltungskomponente weitergegeben. Dadurch erhalten Verwaltungskomponenten eine aggregierte Sicht auf den Ressourcen-Verbrauch der untergeordneter Verwaltungskomponenten. Eine Netz-spezifische Verwaltungskomponente kennt dadurch den tatsächlichen Verbrauch der ihr zugeordneten Protokoll-spezifischen Verwaltungskomponenten bzw. der entsprechenden Dienst-spezifischen Protokolle. Die Knoten-weite Verwaltungskomponente kennt den tatsächlichen Verbrauch der auf dem Endsystem verwendeten Netz-spezifischen Verwaltungskomponenten bzw. der entsprechenden DsNs.

Anpassung der Ressourcen-Zuteilung

Erkennt eine Verwaltungskomponente zur Laufzeit bei der Überwachung des eigenen Ressourcen-Verbrauchs einen höheren Ressourcen-Bedarf, als Ressourcen verfügbar sind, wird eine Anpassung der Ressourcen-Zuteilung durchgeführt. Dabei stehen der Verwaltungskomponente drei Möglichkeiten zur Verfügung: (1) Entziehen von zugeteilten Ressourcen von untergeordneten Verwaltungskomponenten, (2) Anfordern weiterer Ressourcen von übergeordneter Verwaltungskomponente oder (3) Fehlerbehandlung.

(1) In der ersten Möglichkeit wird zunächst versucht, untergeordneten Verwaltungskomponenten nicht benötigte Ressourcen zu entziehen. Durch die Überwachung des Ressourcen-Verbrauchs in den Verwaltungskomponenten und durch den Informationsaustausch zwischen den Verwaltungskomponenten besitzt eine Verwaltungskomponente Informationen über den tatsächlichen Ressourcen-Verbrauch ihrer untergeordneten Verwaltungskomponenten. Ist der tatsächliche Ressourcen-Verbrauch einer untergeordneten Verwaltungskomponente geringer als die ihr zugewiesenen Ressourcen, können ihr Ressourcen über die Schnittstellen-Funktion *withdraw()* entzogen werden. Werden auf diese Weise einer oder mehreren untergeordneten Verwaltungskomponenten ausreichend viele Ressourcen entzogen, um den eigenen Ressourcen-Bedarf abzudecken, kann die Verwaltungskomponente zu ihrem regulären Betrieb zurückkehren. Reichen die dadurch erhaltenen Ressourcen nicht aus, um den eigenen Ressourcen-Bedarf abzudecken, greift die Verwaltungskomponente auf die zweite Möglichkeit zurück.

(2) In der zweiten Möglichkeit wird versucht, die noch benötigten Ressourcen von der übergeordneten Verwaltungskomponente zu erhalten. Hierfür fordert die Verwaltungskomponente mit Hilfe der Schnittstellen-Funktion *request()* die Ressourcen von der übergeordneten Verwaltungskomponente an. Besitzt die übergeordnete Verwaltungskomponente ausreichend Ressourcen, weist sie die Ressourcen zu. Besitzt die übergeordnete Verwaltungskomponente nicht ausreichend viele Ressourcen, kann sie selbst versuchen, über die hier beschriebenen Möglichkeiten die Ressourcen zu erhalten. Erhält die Verwaltungskomponente ausreichend viele Ressourcen von der übergeordneten Verwaltungskomponente, kann sie zu ihrem regulären Betrieb zurückkehren. Reichen die

erhaltenen Ressourcen immer noch nicht aus, um den Ressourcen-Bedarf abzudecken, fährt die Verwaltungskomponente mit der dritten Möglichkeit fort.

(3) Bei der dritten Möglichkeit wird eine Fehlerbehandlung durchgeführt. Dabei wird versucht mit dieser Situation umzugehen. Das genaue Verhalten in dieser Situation ist spezifisch für die Verwaltungskomponente. Eine Verwaltungskomponente kann beispielsweise die noch benötigten Ressourcen untergeordneten Verwaltungskomponenten entziehen, obwohl diese selbst diese Ressourcen benötigen. An diese angepassten Ressourcen-Zuteilungen müssen sich dann die untergeordneten Verwaltungskomponenten anpassen. Dies kann zu Fehlern in den untergeordneten Verwaltungskomponenten führen, die dann selbst wieder aufgelöst werden müssen. Eine Alternative ist selbst zu versuchen, sich an die eingeschränkten Ressourcen anzupassen. Des weiteren ist es möglich, eine Fehlerbeschreibung zu generieren und den Nutzer über die Situation zu informieren.

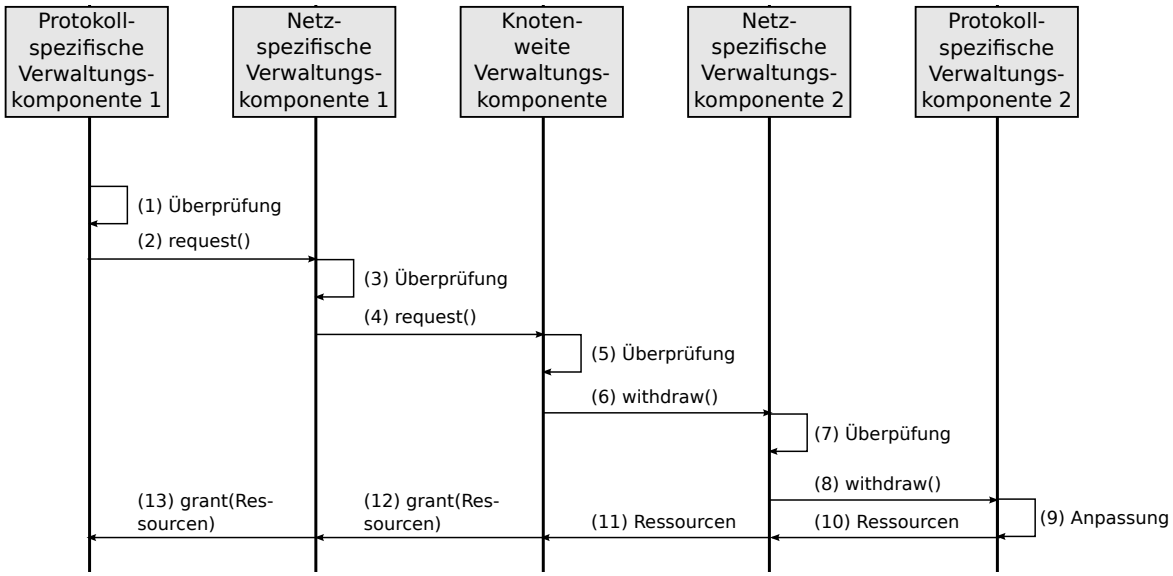

Abbildung 4.11: *Anpassung der Ressourcen-Zuteilung im Hierarchischen Knoten-Management*

In Abbildung 4.11 wird ein Beispiel einer Umverteilung von Ressourcen auf einem Endsystem anhand eines Sequenzdiagrammes gezeigt. Es werden fünf beteiligte Verwaltungskomponenten dargestellt: *Protokoll-spezifische Verwaltungskomponente 1* (PsVK 1), *Netz-spezifische Verwaltungskomponente 1* (NsVK 1), die *Knoten-weite Verwaltungskomponente*, *Netz-spezifische Verwaltungskomponente 2* (NsVK 2) und *Protokoll-spezifische Verwaltungskomponente 2* (PsVK 2). Die Interaktion zwischen den Verwaltungskomponenten wird mit Hilfe von Pfeilen und den verwendeten Funktionen dargestellt.

(1) Die *PsVK 1* überprüft ihren Ressourcen-Verbrauch (*Überprüfung*), und stellt fest, dass sie mehr Ressourcen benötigt als ihr zur Verfügung stehen. (2) Daher fordert sie

weitere Ressourcen von der *NsVK 1* an (*request()*). (3) Die *NsVK 1* überprüft, ob ihr genug Ressourcen zur Verfügung stehen, um die Anfrage zu erfüllen (*Überprüfung*). Dabei stellt sie fest, dass ihr nicht genug Ressourcen vorliegen. (4) Daher fordert sie weitere Ressourcen von der *Knoten-weiten Verwaltungskomponente* an (*request()*). (5) Die *Knoten-weite Verwaltungskomponente* überprüft, ob ihr genug Ressourcen zur Verfügung stehen, um die Anfrage zu erfüllen (*Überprüfung*). Dabei stellt sie fest, dass ihr nicht genug Ressourcen vorliegen. (6) Daher entzieht sie der *NsVK 2* die entsprechenden Ressourcen (*withdraw()*). (7) Die *NsVK 2* überprüft, wie sie ihren eigenen Ressourcen-Bedarf anpassen kann, um die Anfrage zu erfüllen (*Überprüfung*). (8) Um die benötigten Ressourcen zu erhalten, beschließt sie, der *PsVK 2* die Ressourcen zu entziehen (*withdraw()*). (9) Die *PsVK 2* passt ihren eigenen Ressourcen-Bedarf an, um die Anfrage zu erfüllen (*Anpassung*) und (10) gibt die *Ressourcen* zurück. (11) Die *NsVK 2* übergibt die *Ressourcen* der Knoten-weiten Verwaltungskomponente. (12) Die Knoten-weite Verwaltungskomponente übergibt die Ressourcen an die *NsVK 1* (*grant(Ressourcen)*). (13) Diese übergibt die Ressourcen an die *PsVK 1* (*grant(Ressourcen)*). Diese speichert die Ressourcen und kann sie schließlich verwenden.

4.6.4 Entfernen Dienst-spezifischer Netze

Wird eine Verwaltungskomponente aus dem HKM entfernt, meldet sich diese von der Verwaltungskomponente, die sich über ihr in der Hierarchie befindet, ab. Dadurch weiß die übergeordnete Verwaltungskomponente, dass die untergeordnete Verwaltungskomponente nicht mehr existiert und kann dessen Ressourcen freigeben.

Wenn ein Endsystem nicht mehr auf ein DsN zugreift, kann es die Verbindung zu diesem DsN abbauen. Dabei beendet das Endsystem die Virtuelle Verbindung zu dem DsN und entfernt den entsprechenden Dienst-spezifischen Protokoll-Stapel bzw. die entsprechenden Dienst-spezifischen Protokolle. Dadurch werden auch die entsprechenden Verwaltungskomponenten entfernt. Dabei werden zunächst die Protokoll-spezifischen Verwaltungskomponenten und anschließend die Netz-spezifische Verwaltungskomponente entfernt. Wird eine Protokoll-spezifische Verwaltungskomponente entfernt, meldet sie sich von der Netz-spezifischen Verwaltungskomponente ab. Dadurch weiß die Netz-spezifische Verwaltungskomponente, dass die Protokoll-spezifische Verwaltungskomponente nicht mehr existiert und gibt dessen Ressourcen wieder frei. Haben sich alle Protokoll-spezifischen Verwaltungskomponenten bei der Netz-spezifischen Verwaltungskomponente abgemeldet, erhält die Netz-spezifische Verwaltungskomponente somit auch alle Ressourcen wieder zurück und weiß, dass sie selbst entfernt werden kann. Wird die Netz-spezifische Verwaltungskomponente entfernt, meldet sie sich bei der Knoten-weiten Verwaltungskomponente ab. Hierdurch weiß die Knoten-weite Verwaltungskomponente, dass die Netz-spezifische Verwaltungskomponente nicht mehr existiert. Indem die Knoten-weite Verwaltungskomponente die Ressourcen-Zuteilungen der Netz-spezifischen

Verwaltungskomponente entfernt, stehen die Ressourcen wieder anderen Verwaltungskomponenten zur Verfügung.

4.6.5 Zusammenfassung

Die Ressourcen-Verwaltung im HKM erlaubt das Zuteilen der auf dem Endsystem verfügbaren Ressourcen an die Komponenten von DsNs. Außerdem überwacht die Ressourcen-Verwaltung den tatsächlichen Ressourcen-Verbrauch der verschiedenen Komponenten zur Laufzeit. Benötigt eine Komponente mehr Ressourcen, als ihr zugeteilt ist, können die Ressourcen-Zuteilungen auf dem Endsystem umverteilt werden. Dabei können anderen Komponenten Ressourcen entzogen und der entsprechenden Komponente zugewiesen werden. Auf sich ändernden Ressourcen-Zuteilen können die Komponenten durch Komponenten-spezifische Anpassungen reagieren.

4.7 Implementierung

Für die Evaluierung wurde ein Prototyp des HKM implementiert. Dazu wurden die beiden Rahmenwerke NENA und Distack [34] erweitert und miteinander kombiniert. Ein Überblick über die Implementierung des HKM wird in Abbildung 4.12 dargestellt.

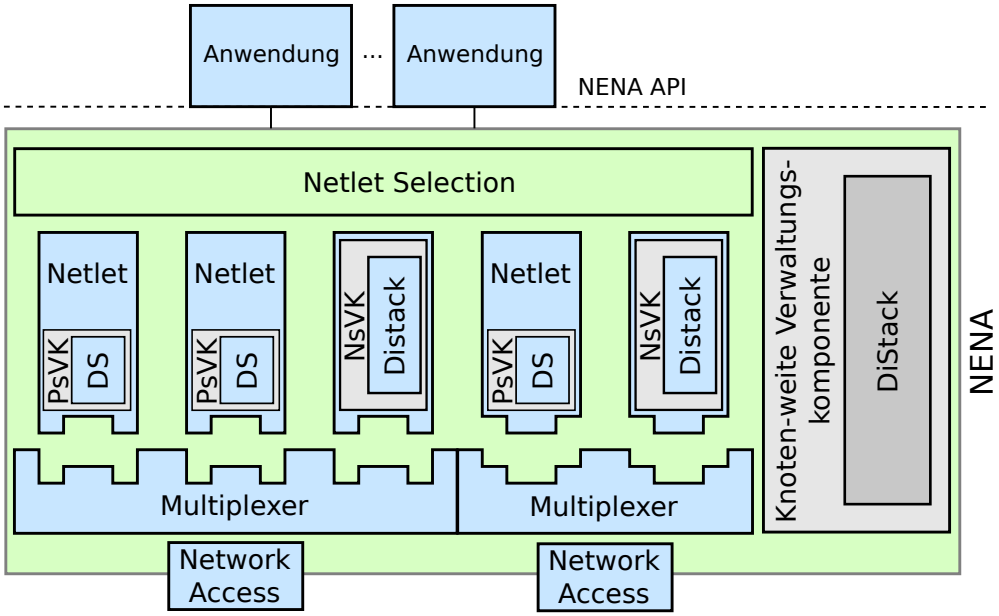

Abbildung 4.12: *Implementierung des Hierarchischen Knoten-Managements in NENA*

In der Abbildung wird NENA, das in Abschnitt 5.2 näher vorgestellte Rahmenwerk für DsNs, zusammen mit den verschiedenen Verwaltungskomponenten schematisch darge-

stellt. NENA greift auf zwei DsNs zu. Jedes DsN verwendet eine Virtuelle Verbindung, in NENA *Network Access* genannt. Für die DsNs betreibt NENA *Netlets*, Container für Dienst-spezifische Protokolle. Die Netlets eines DsN greifen über einen gemeinsamen *Multiplexer* auf die Virtuelle Verbindung zu. Zwei Anwendungen greifen auf NENA zu, die über die *Netlet Selection* den beiden DsNs zugewiesen werden. In NENA wurden die Verwaltungskomponenten integriert. *PsVK* steht für Protokoll-spezifische Verwaltungskomponente, *NsVK* für Netz-spezifische Verwaltungskomponente. Der Komponenten-spezifische Management-Zyklus in den verschiedenen Verwaltungskomponenten wurde aufgrund der Flexibilität und vereinfachten Implementierung mit Hilfe von Distack-Instanzen implementiert (siehe Abschnitt 4.7.2). Diese werden als *Distack* oder *DS* in der Abbildung dargestellt.

Im Folgenden werden die Verwaltungskomponenten in NENA näher beschrieben. Anschließend wird darauf eingegangen, wie der Management-Zyklus in den Verwaltungskomponenten mit Hilfe von Distack implementiert wurde.

4.7.1 Verwaltungskomponenten

Die Protokoll-spezifischen Verwaltungskomponenten werden mit den entsprechenden Dienst-spezifischen Protokollen gekoppelt und in gemeinsamen Netlets zusammengefasst. Hierdurch sind sie für die Verwaltung der entsprechenden Netlets verantwortlich.

Die Netz-spezifischen Verwaltungskomponenten werden in NENA als separate Netlets implementiert. Ein solches Netlet wird über einen gemeinsamen Multiplexer mit den anderen Netlets des selben DsN gruppiert. Eine Netz-spezifische Verwaltungskomponente greift auf den Multiplexer, die Virtuellen Verbindungen und die Netlets des entsprechenden DsN zu und ist für deren Verwaltung verantwortlich.

Die Knoten-weite Verwaltungskomponente wird als zusätzliche Komponente in NENA integriert. Diese Komponente wird beim Start von NENA initialisiert und ausgeführt. Die Knoten-weite Verwaltungskomponente verwaltet Netz-spezifische Verwaltungskomponenten mit Hilfe der entsprechenden Netlets.

4.7.2 Management-Zyklus

Für die Implementierung des Management-Zyklus innerhalb der Verwaltungskomponenten wird Distack verwendet. Distack ist ein modulares Management-Rahmenwerk. Es wurde ursprünglich für die Anomalie-basierte Angriffs- und Fehlererkennung in Netzen entworfen. Es erlaubt eine schrittweise Überwachung von Netz-Eigenschaften oder Protokoll-Parametern. Dabei sind die ersten Schritte der Überwachung in der Regel so entworfen, dass sie effizient durchgeführt werden können. Ergibt sich ein Verdacht auf eine Fehler- oder Angriffssituation, werden weitere Schritte durchgeführt. In diesen Schritten werden die Überprüfungen inkrementell komplexer. Erkennt ein Schritt durch

seine Überprüfungen, dass doch kein Angriff oder Fehler vorliegt, wird die Überprüfung beendet. Erkennt ein Schritt, dass definitiv ein Fehler oder ein Angriff vorliegt, werden Mechanismen zum Beheben der Situation durchgeführt. Dies erlaubt eine flexible Durchführung des Management-Zyklus bestehend aus Monitoring, Entscheidungsfindung und Anpassung und vereinfacht damit die Implementierung der Verwaltungskomponenten.

Für den Prototypen des HKM wird der Management-Zyklus in den Verwaltungskomponenten mit Hilfe von Distack-Instanzen implementiert. Jede Verwaltungskomponente erhält eine eigene Distack-Instanz und führt einen eigenständigen Management-Zyklus auf Basis von Distack-Modulen durch, die an die entsprechende Verwaltungskomponente angepasst sind. Für die Evaluierung wurden daher auch Distack-Module implementiert.

4.8 Evaluierung

In diesem Abschnitt wird das HKM mit Hilfe des zuvor beschriebenen Prototypen evaluiert. Dabei wird die Funktionsfähigkeit des HKM und der Ressourcen-Verwaltung gezeigt. Zunächst wird der Versuchsaufbau gezeigt. Anschließend werden die beiden Testfälle vorgestellt. Im ersten Testfall wird die Anpassung an Netz-Eigenschaften untersucht. Im zweiten Testfall wird die Ressourcen-Verwaltung untersucht. Abschließend werden die Ergebnisse der Evaluation zusammengefasst.

4.8.1 Versuchsaufbau

Für die Evaluierung des HKM wurde der in Abbildung 4.13 dargestellte Versuchsaufbau verwendet. Es wurde ein Anwendungsfall betrachtet, in dem ein Endsystem zwei DsNs verwendet, um darüber Video-Daten an andere Endsysteme zu übertragen (Video-Streaming).

Der Versuchsaufbau besteht aus drei Endsystemen. Ein Endsystem ist der *Sender*, auf dem die zwei Anwendungen *Video-Server 1* (*VS1*) und *Video-Server 2* (*VS2*) ausgeführt werden. Der Sender verwendet zwei DsNs. Die Anwendung *VS1* ist dem ersten DsN, *Dienst-spezifisches Netz 1*, zugeordnet. Die Anwendung *VS2* ist dem zweiten DsN, *Dienst-spezifisches Netz 2*, zugeordnet. Für die beiden DsNs unterhält der Sender jeweils Virtuelle Verbindungen zu den anderen beiden Endsystemen *Empfänger 1* und *Empfänger 2*. Der Sender ist mit *Empfänger 1* über *DsN 1* und mit *Empfänger 2* über *DsN 2* verbunden. Auf dem *Empfänger 1* wird die Anwendung *Video-Client 1* (*VC1*) und auf dem *Empfänger 2* die Anwendung *Video-Client 2* (*VC2*) ausgeführt.

Die verwendeten Anwendungen sind Anwendungen zur Übertragung und zum Abspielen von Videos. *VS1* und *VS2* sind die Server-Anwendungen, die den Empfängern Videos bereitstellen. Hierfür sendet jede der Server-Anwendungen Video-Daten über das entsprechende DsN an die entsprechende Client-Anwendung. *VC1* und *VC2* sind die

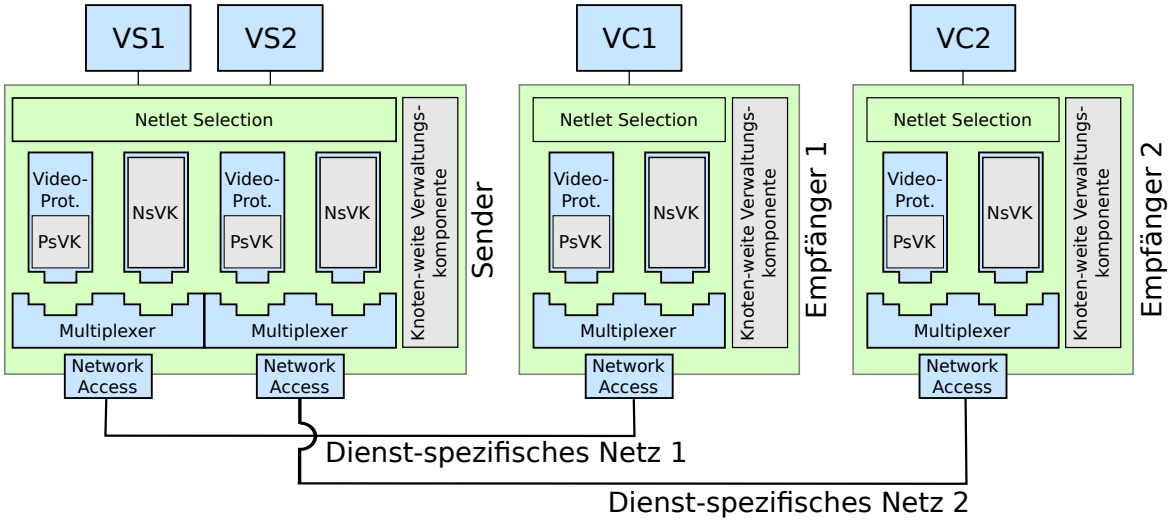

Abbildung 4.13: *Versuchsaufbau für die Evaluierung des Hierarchischen Knoten-Managements*

Client-Anwendungen, die jeweils auf die Server-Anwendungen zugreifen, um ein Video zu empfangen und abzuspielen. Eine Client-Anwendung empfängt über das entsprechende DsN die Video-Daten und spielt sie ab.

Da es sich um Video-Streaming Anwendungen handelt, sind die verwendeten DsNs bzw. verwendeten Dienst-spezifischen Protokolle auf die Übertragung von Video-Daten ausgelegt. Jedes DsN verwendet ein Dienst-spezifisches Protokoll (*Video-Prot.*) zur Übertragung von Video-Daten. Um mit bei der Übertragung über das Netz auftretenden Fehlern umzugehen, implementiert das Video-Streaming Protokoll einen Forward-Error-Correction (FEC) Mechanismus. Der FEC-Mechanismus kann über einen Parameter zur Laufzeit so angepasst werden, dass die übertragenen Redundanzen in drei Stufen erhöht und verringert werden können. Die erste Stufe bedeutet, dass keine Redundanzen übertragen werden. Die zweite Stufe bedeutet, dass jedes Paket doppelt übertragen wird. Die dritte Stufe bedeutet, dass jedes Paket drei mal übertragen wird.

Auf den Endsystemen wird das HKM ausgeführt. Die Protokoll-spezifischen Verwaltungskomponenten (*PsVK*) überwachen den bei der Übertragung der Video-Daten auftretenden Informationsverlust. Außerdem sind sie in der Lage, den Parameter des FEC-Mechanismus bzw. die FEC-Stufe des Dienst-spezifischen Protokolls zu verändern. Durch die Überwachung des Informationsverlusts und die Veränderung der FEC-Stufe sind die Protokoll-spezifischen Verwaltungskomponenten daher auch in der Lage, die FEC-Stufe an die Netz-Eigenschaften anzupassen. Außerdem sind ihnen die für die Übertragung mit den unterschiedlichen FEC-Stufen benötigten Ressourcen bekannt. Die benötigten Ressourcen fordern sie von den Netz-spezifischen Verwaltungskomponenten (*NsVK*) an.

Die Netz-spezifischen Verwaltungskomponenten und die *Knoten-weiten Verwaltungskomponenten* verwalten die in den DsNs und auf den Endsystemen verfügbaren Ressourcen.

4.8.2 Anpassung an Netz-Eigenschaften

In diesem Experiment wird die Anpassung an die Eigenschaften der Netz-Infrastruktur zur Laufzeit mit Hilfe des HKM untersucht. Hierfür werden Video-Daten über eine verlustbehaftete Übertragungsstrecke zwischen dem Sender und einem Empfänger übertragen. Die FEC-Stufe wird dabei angepasst, um die Übertragungsfehler im Empfänger zu kompensieren.

Für dieses Experiment werden der Sender und ein Empfänger, *Empfänger 1*, verwendet. Außerdem wird nur das DsN, über das die beiden Knoten verbunden sind, verwendet (*DsN 1*). Der Sender überträgt die Video-Daten an den Empfänger über das DsN und das Dienst-spezifische Protokoll (Video-Streaming Protokoll). Die für das DsN verwendete Übertragungsstrecke ist verlustbehaftet. Dies bedeutet, dass Paketverluste bei der Übertragung auftreten können. Andere negative Effekte, wie beispielsweise Paketvertauschungen, werden in diesem Experiment nicht betrachtet. Die Paketverlustrate variiert in diesem Experiment zwischen 0 %, 5 % und 15 %.

Das Video-Streaming Protokoll versieht die übertragenen Pakete mit Sequenznummern. Hierdurch ist eine Erkennung von Paketverlusten möglich. Der FEC-Mechanismus des Video-Streaming Protokolls erlaubt, die selben Video-Daten als Duplikate in verschiedenen Paketen mit unterschiedlichen Sequenznummern zu übertragen. Die in den Paketen übertragenen Video-Daten werden mit eigenen Sequenznummern versehen. Hierdurch ist es möglich zu erkennen, ob Video-Daten beim Empfänger (trotz Paketverlusten) ankommen. Dies ist wichtig, um zu erkennen, ob die aktuelle FEC-Stufe ausreicht, die Paketverluste auszugleichen. Die Verluste von Video-Daten werden in diesem Experiment als Informationsverlust bezeichnet. Die FEC-Stufe des Video-Streaming Protokolls wird erhöht, wenn der Informationsverlust einen (vom Nutzer gesetzten) Schwellenwert überschreitet. Dieser Schwellenwert beträgt in diesem Experiment 2%. Verringert sich die Paketverlustrate, kann die FEC-Stufe wieder verringert werden. Da bei dem verwendeten FEC-Mechanismus Video-Daten mehrfach übertragen werden, werden die Duplikate der Video-Daten beim Empfänger verworfen, bevor die Video-Daten an die Anwendung weitergegeben werden.

In diesem Experiment wird nur das Verhalten des Senders betrachtet. Signalisierung zwischen dem Empfänger und dem Sender, um den Informationsverlust zu signalisieren, oder die Reaktionen auf *Empfänger 1* werden nicht näher betrachtet. Es wurden 100 Versuchsläufe durchgeführt.

Der zeitliche Verlauf des Experiments wird in Abbildung 4.14 dargestellt. Auf der x-Achse wird der zeitliche Verlauf bzw. der aktuelle Zeitpunkt des Experiments in Sekunden dargestellt. Auf der linken y-Achse wird die FEC-Stufe des Video-Streaming-Protokolls

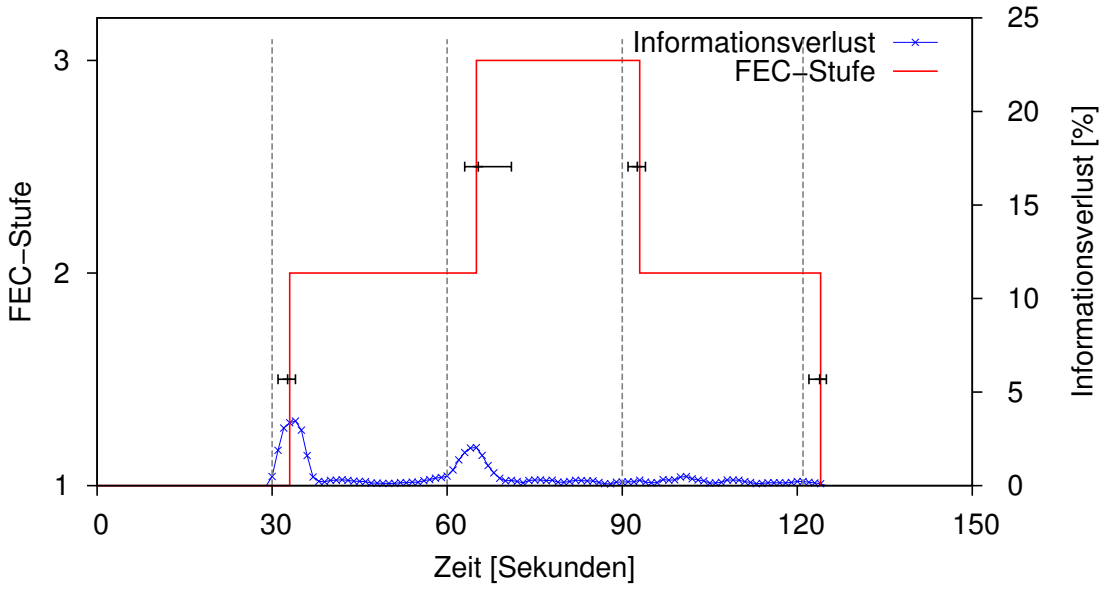

Abbildung 4.14: *Evaluierung der Anpassung an Netz-Eigenschaften mit dem Hierarchischen Knoten-Management*

angezeigt. Auf der rechten y-Achse wird der in der Verwaltungskomponente des Video-Streaming-Protokolls gemessene Informationsverlust bei der Übertragung in Prozent dargestellt. Die Kurve *Informationsverlust* stellt den Informationsverlust über die Zeit in einem exemplarisch ausgewählten Versuchslauf dar. *FEC-Stufe* stellt den Verlauf der FEC-Stufe des Video-Streaming Protokolls über die Zeit in dem selben ausgewählten Versuchslauf dar. Die horizontalen Fehler-Balken stellen die frühesten, mittleren (arithmetisches Mittel) und spätesten Zeitpunkte dar, an denen in diesem Experiment in 100 Versuchsläufen die FEC-Stufe angepasst worden ist.

Das Experiment startet zum Zeitpunkt 0. Der Sender und der Empfänger (*Empfänger 1*) werden gestartet. Dabei werden alle Komponenten auf den Endsystemen initialisiert. Zum Zeitpunkt 30 fängt der Sender an, Video-Daten an den Empfänger zu übertragen. Außerdem beträgt die Paketverlustrate auf der Übertragungsstrecke 5 %. Daher werden bei der Übertragung nicht alle Pakete vom Empfänger empfangen. Der Empfänger erkennt die Paketverluste und den damit verbundenen Informationsverlust und signalisiert diese Informationen an den Sender. Der gemessene Informationsverlust steigt und überschreitet den Schwellenwert. Der Sender sendet daher die Video-Daten mit einer höheren FEC-Stufe zum Empfänger. Dadurch wird der Informationsverlust reduziert. Zum Zeitpunkt 60 erhöht sich die Paketverlustrate von 5 % auf 15 %. Dadurch erhöht sich wiederum der Informationsverlust. Der Sender erhöht daher die FEC-Stufe. Dadurch senkt sich erneut der Informationsverlust. Zum Zeitpunkt 90 verbessert sich die Paketverlustrate von 15 % auf 5 %. Der beim Empfänger erkannte Paketverlust sinkt daher unter einen

Schwellenwert, der den Sender dazu veranlasst, die FEC-Stufe zu senken. Da sich der Informationsverlust nicht erhöht, wird diese FEC-Stufe auch beibehalten. Zum Zeitpunkt 120 senkt sich die Paketverlustrate weiter von 5 % auf 0 %, woraufhin der Sender die FEC-Stufe weiter absenkt. Anschließend wird das Experiment beendet.

Dieses Experiment zeigt, dass das Monitoring und die Anpassung an die Netz-Eigenschaften mit Hilfe des HKM in einer Verwaltungskomponente funktioniert. Die FEC-Stufe des Video-Streaming Protokolls wird erfolgreich zur Laufzeit an die im Netz auftretende Paketverlustrate angepasst. Dabei wird der Informationsverlust unter dem gesetzten Schwellenwert gehalten. Außerdem werden die FEC-Stufe und damit die verbrauchten Netz-Ressourcen auf dem tatsächlich benötigten Wert gehalten. Aufbauend darauf wird in der folgenden Evaluation das Zusammenspiel mehrerer Verwaltungskomponenten in allen Hierarchie-Ebenen zusammen mit der Ressourcen-Verwaltung untersucht.

4.8.3 Ressourcen-Verwaltung

In diesem Experiment wird die Ressourcen-Verwaltung im HKM untersucht. Hierfür versendet der Sender zwei Videos an zwei Empfänger über zwei unterschiedliche DsNs. Diese beiden DsNs konkurrieren auf dem Sender um die zur Verfügung stehenden Netz-Ressourcen. Außerdem werden, wie im vorherigen Experiment, die Video-Daten über eine verlustbehaftete Übertragungsstrecke zwischen dem Sender und den Empfängern übertragen. Die FEC-Stufe in den Video-Streaming Protokollen wird angepasst, um die Übertragungsfehler in den Empfängern zu kompensieren.

Für dieses Experiment werden der Sender und die beiden Empfänger *Empfänger 1* und *Empfänger 2* verwendet. Außerdem werden beide DsNs, *DsN 1* und *DsN 2* verwendet. Der Sender versendet die Video-Daten von Anwendung *VS1* an *Empfänger 1* über das *DsN 1* und das entsprechende Dienst-spezifische Protokoll (Video-Streaming Protokoll). Die Video-Daten von Anwendung *VS2* werden an *Empfänger 2* über das *DsN 2* und das entsprechende Dienst-spezifische Protokoll versendet. Die für das DsN verwendete Übertragungsstrecke ist verlustbehaftet. Dies bedeutet, dass Paketverluste bei der Übertragung auftreten können. Andere negative Effekte, wie beispielsweise Paketvertauschungen, werden in diesem Experiment nicht betrachtet. Die Paketverlustrate variiert zwischen 0 % und 15 %.

Die verwendeten Video-Streaming Protokolle und deren FEC-Mechanismen entsprechen den im vorherigen Experiment verwendeten. Allerdings teilen sich die beiden verwendeten DsNs auf dem Sender eine gemeinsame Netzwerkschnittstelle. Dadurch konkurrieren sie um die verfügbare Datenrate dieser Netzwerkschnittstelle. Je höher die verwendete FEC-Stufe eines Video-Streaming Protokolls ist, desto höher ist die benötigte Datenrate des Protokolls. Die verfügbare Datenrate ist zudem beschränkt. Sie reicht aus, um die Video-Daten über das DsN 1 oder die Video-Daten über das DsN 2 mit der maximalen FEC-Stufe zu übertragen. Allerdings reichen die Netz-Ressourcen nicht aus,

um über beide DsNs die Video-Daten mit eingeschalteter FEC zu übertragen. Außerdem hat der Nutzer in diesem Experiment dem HKM eine Richtlinie übergeben, dass ihm die Nutzung des *DsN 2* wichtiger ist als die Nutzung des *DsN 1*.

In diesem Experiment wird nur das Verhalten des Senders betrachtet. Signalisierung zwischen dem Sender und *Empfänger 1* oder *Empfänger 2*, um den Informationsverlust zu signalisieren, oder die Reaktionen auf *Empfänger 1* und *Empfänger 2* werden nicht näher betrachtet. Es wurden 100 Versuchsläufe durchgeführt.

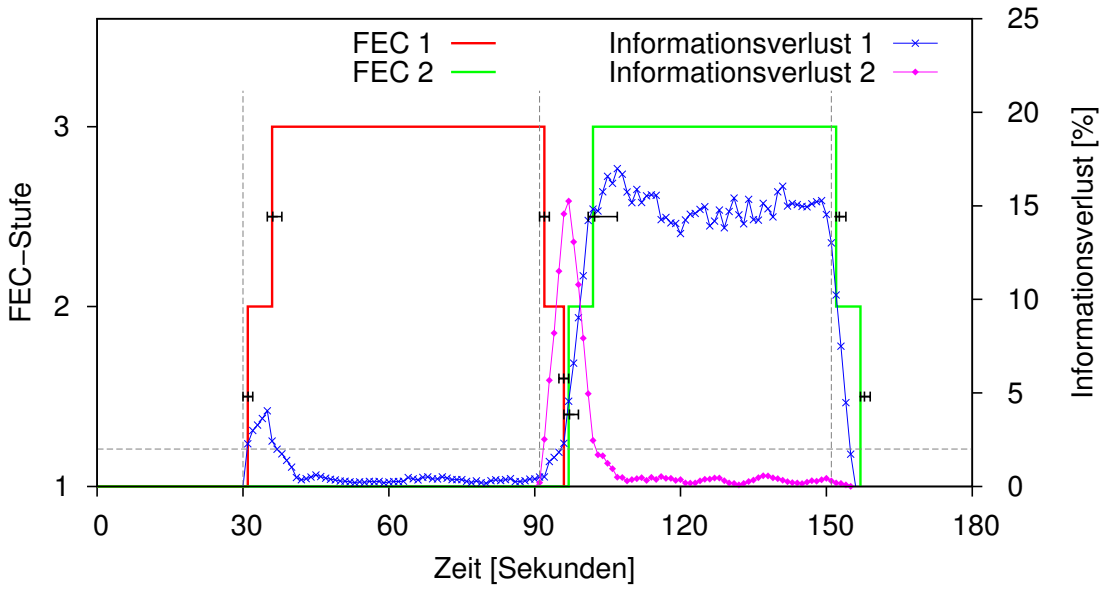

Abbildung 4.15: *Evaluierung der Ressourcen-Verwaltung mit dem Hierarchischen Knoten-Management*

Der zeitliche Verlauf des Experiments wird in Abbildung 4.15 dargestellt. Auf der x-Achse wird der zeitliche Verlauf bzw. der aktuelle Zeitpunkt des Experiments in Sekunden dargestellt. Auf der linken y-Achse wird die FEC-Stufe der Video-Streaming-Protokolle angezeigt. Auf der rechten y-Achse wird der in den Verwaltungskomponenten der Video-Streaming Protokolle gemessene Informationsverlust bei der Übertragung in Prozent dargestellt. *Informationsverlust 1* stellt den Informationsverlust im *DsN 1* über die Zeit in einem exemplarisch ausgewählten Versuchslauf dar. *Informationsverlust 2* stellt den Informationsverlust im *DsN 2* über die Zeit in dem selben ausgewählten Versuchslauf dar. *FEC 1* stellt den Verlauf der FEC-Stufe des Video-Streaming Protokolls im *DsN 1* über die Zeit in dem selben Versuchslauf dar. *FEC 2* stellt den Verlauf der FEC-Stufe des Video-Streaming Protokolls im *DsN 2* über die Zeit im selben Versuchslauf dar. Die horizontalen Fehler-Balken stellen die frühesten, mittleren (arithmetisches Mittel) und spätesten Zeitpunkte dar, an denen in diesem Experiment in 100 Versuchsläufen die FEC-Stufen der beiden Video-Streaming Protokolle angepasst worden sind.

Das Experiment startet zum Zeitpunkt 0. Der Sender und die beiden Empfänger (*Empfänger 1* und *Empfänger 2*) werden gestartet. Dabei werden alle Komponenten auf den Endsystemen initialisiert. Zum Zeitpunkt 30 fängt der Sender an, Video-Daten an *Empfänger 1* zu übertragen. Außerdem beträgt die Paketverlustrate auf der Übertragungsstrecke 15 %. Der bei der Übertragung gemessene Informationsverlust steigt und überschreitet den Schwellenwert. Der Sender versendet daher die Video-Daten mit einer höheren FEC-Stufe. Da dadurch der Informationsverlust nicht unter den Schwellenwert sinkt, erhöht der Sender erneut die FEC-Stufe. Dadurch wird der Informationsverlust reduziert. Bei der Erhöhung der FEC-Stufe werden der Protokoll-spezifischen Verwaltungskomponente durch die Ressourcen-Verwaltung die Netz-Ressourcen zugeteilt, die für die Datenübertragung mit der höchsten FEC-Stufe benötigt werden. Hierdurch sind keine weiteren Netz-Ressourcen auf dem Sender verfügbar.

Zum Zeitpunkt 90 fängt der Sender an, Video-Daten an *Empfänger 2* zu übertragen. Auch hier beträgt die Paketverlustrate auf der Übertragungsstrecke 15 %. Der bei der Übertragung gemessene Informationsverlust steigt und überschreitet den Schwellenwert. Der Sender muss daher die Video-Daten an den Empfänger mit einer höheren FEC-Stufe senden. Allerdings reichen die verfügbaren Netz-Ressourcen in der Protokoll-spezifischen Verwaltungskomponente hierfür nicht aus. Daher fordert sie mehr Ressourcen von der Netz-spezifischen Verwaltungskomponente an. Da diese auch nicht die benötigten Ressourcen besitzt, fordert diese die Ressourcen von der Knoten-weiten Verwaltungskomponente an. Da alle Netz-Ressourcen der Netzwerkschnittstelle vergeben sind, entscheidet die Netz-spezifische Verwaltungskomponente aufgrund der Nutzer-Richtlinie, dass die Ressourcen dem DsN 1 entzogen werden. Darüber informiert sie die entsprechende Netz-spezifische Verwaltungskomponente, die wiederum die Ressourcen der Protokoll-spezifischen Verwaltungskomponente entzieht. Um die Ressourcen freizugeben, reduziert die Protokoll-spezifische Verwaltungskomponente des Video-Streaming Protokolls im DsN 1 die FEC-Stufe schrittweise, bis genug Ressourcen frei werden. Hierdurch wird der FEC-Mechanismus in dem Video-Streaming Protokoll abgeschaltet und der entsprechende Informationsverlust steigt auf ca. 15%, was der Paketverlustrate der Übertragungsstrecke entspricht. Die freigegebenen Ressourcen werden entlang der Hierarchie an die Knoten-weite Verwaltungskomponente weiter gegeben. Diese teilt die Ressourcen dem DsN 2 zu. Die Ressourcen werden daraufhin entlang der Hierarchie weiter gegeben, bis die Protokoll-spezifische Verwaltungskomponente im DsN 2 sie schließlich erhält. Die Protokoll-spezifische Verwaltungskomponente erhöht die FEC-Stufe zunächst auf Stufe 1. Da dadurch der Informationsverlust nicht unter den Schwellenwert sinkt, erhöht der Sender erneut die FEC-Stufe. Dadurch wird der Informationsverlust schließlich reduziert.

Zum Zeitpunkt 150 sinkt für beide DsNs die Paketverlustrate auf 0 %. Daher wird im DsN 2 schrittweise die FEC-Stufe des Video-Streaming Protokolls verringert. Anschließend wird das Experiment beendet.

Dieses Experiment zeigt, dass die Ressourcen-Verwaltung im HKM funktioniert. Ressourcen werden von den Verwaltungskomponenten überwacht. Benötigt eine Verwaltungskomponente mehr Ressourcen, kann sie diese anfordern. Darüber hinaus können den Verwaltungskomponenten Ressourcen entzogen werden. Hierdurch werden die Ressourcen auf dem Endsystem erfolgreich reguliert und umverteilt.

4.8.4 Zusammenfassung

Die Evaluierung des HKM zeigt dessen Funktionsfähigkeit anhand eines Prototypen. Es wurde gezeigt, dass das HKM in der Lage ist, Netz-Eigenschaften zu überwachen und Komponenten an die sich ändernden Netz-Eigenschaften zur Laufzeit anzupassen. Darüber hinaus wurde gezeigt, dass das HKM eine Verwaltung der Ressourcen des Endsystems erlaubt. Es wurde gezeigt, dass den Verwaltungskomponenten Ressourcen zur Laufzeit zugewiesen und entzogen werden können und somit eine Regulierung des Ressourcen-Verbrauchs der Komponenten auf dem Endsystem erreicht wird.

4.9 Zusammenfassung

In diesem Kapitel wurde ein Hierarchisches Knoten-Management für DsNs vorgestellt. Das HKM erlaubt, die auf dem Endsystem verwendeten DsNs bzw. Dienst-spezifischen Protokolle zur Laufzeit an die Netz-Eigenschaften und die verfügbaren Ressourcen anzupassen. Außerdem erlaubt es eine flexible Ressourcen-Verwaltung. Die auf dem Endsystem verfügbaren Prozessor-, Speicher- und Netz-Ressourcen können zur Laufzeit den DsNs und Dienst-spezifischen Protokollen zugewiesen und entzogen werden. Dem Nutzer wird ermöglicht, das HKM und die Ressourcen-Verwaltung durch Richtlinien zu beeinflussen. Der Nutzer kann mit Hilfe von Richtlinien beispielsweise den genutzten DsNs oder Dienst-spezifischen Protokollen Prioritäten geben, wodurch sie bei der Verteilung der verfügbaren Ressourcen bevorzugt werden. Durch den modularen Aufbau und die Hierarchie erreicht das HKM Flexibilität und kann mit heterogenen DsNs und Dienst-spezifischen Protokollen umgehen. Ändert sich die Menge der DsNs, auf die das Endsystem zugreift, können die entsprechenden Verwaltungskomponenten einfach entfernt oder hinzugefügt werden, ohne das gesamte Verwaltungssystem anpassen zu müssen. Da die Verwaltungskomponenten für die DsNs und Protokolle entworfen werden, erlauben die Verwaltungskomponenten eine an deren Anforderungen und Eigenschaften angepasste Verwaltung. In der Evaluation wurde gezeigt, dass das HKM funktioniert und in der Lage ist, sowohl die Komponenten der DsNs an die Eigenschaften der Netz-Infrastruktur anzupassen, als auch die beschränkten Ressourcen des Endsystems zu verwalten.

Kapitel 5

Hochleistungskommunikation in Dienst-spezifischen Netzen

In diesem Kapitel werden die Grundlagen und Analyse für Kapitel 6 vorgestellt. In Abschnitt 5.1 wird auf den Begriff der Hochleistungskommunikation eingegangen. In Abschnitt 5.2 wird das ursprüngliche NENA, ein Rahmenwerk für Dienst-spezifische Netze (DsN), und der Datenfluss durch ein Endsystem vorgestellt. Aufbauend darauf werden in Abschnitt 5.3 Möglichkeiten für einen schnellen Datenaustausch über ein Rahmenwerk für DsNs vorgestellt. Außerdem werden in Abschnitt 5.4 verschiedene Lösungen präsentiert und miteinander verglichen, die einen direkten Zugriff auf Netzwerkschnittstellen und somit einen schnellen Datenaustausch mit dem Netz erlauben.

5.1 Hochleistungskommunikation

Ein Ziel beim Entwurf des Rahmenwerkes für DsNs in Kapitel 6 ist das Erreichen einer hohen Leistung. Im Rahmen dieser Arbeit bezieht sich der Begriff Hochleistungskommunikation auf einen schnellen Datenaustausch zwischen Endsystemen bzw. Anwendungen auf diesen und dem Netz. Unter schnellem Datenaustausch wird ein Datenaustausch mit einer hohen Paketrate verstanden. Das Ziel beim schnellen Datenaustausch ist das Erreichen der vollen Datenrate auf einem Endsystem auch bei kleinen Paketlängen. In diesem Abschnitt wird zunächst festgelegt, welche Datenraten auf einem Endsystem angenommen werden. Basierend hierauf wird gezeigt, welche Paketraten auf Endsystemen mit aktuellen Netzwerkschnittstellen möglich sind. Anhand existierender Arbeiten wird gezeigt, dass gerade Pakete mit kleiner Paketlänge, bei denen hohe Paketraten auftreten können, häufig genutzt werden und daher effizient in Endsystemen verarbeitet werden müssen, um eine hohe Leistung erreichen zu können.

Die möglichen Datenraten von Netzwerkschnittstellen bei Ethernet [46] haben sich im Laufe der Jahre von Datenraten von 1 Megabit/s bis hin zu Datenraten von 100 Gigabit/s weiterentwickelt. Zum Zeitpunkt der Erstellung dieser Arbeit sind Netzwerkschnittstellen nach dem 10 Gigabit Ethernet Standard [47] zunehmend verbreitet. Die maximalen Datenraten dieser Netzwerkschnittstellen liegen bei 10 Gigabit/s. Die möglichen Paketraten in Abhängigkeit von der Paketlänge errechnen sich nach folgender Formel und werden zusätzlich in Abbildung 5.1 dargestellt.

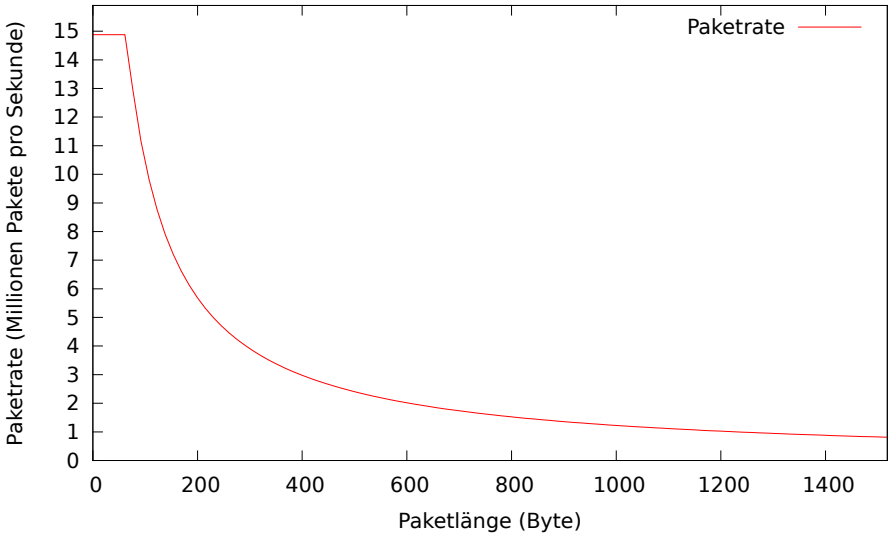

Abbildung 5.1: *Paketraten von 10 Gigabit/s Ethernet abhängig von der Paketlänge.*

$$Paketrate(Paketlänge) = \begin{cases} \dfrac{10000000000\,\frac{Bit}{s}/(8\,\frac{Bit}{Byte})}{20\,Byte + 64\,Byte}, & Paketlänge \leq 64\,Byte \\ \dfrac{10000000000\,\frac{Bit}{s}/(8\,\frac{Bit}{Byte})}{20\,Byte + Paketlänge}, & 64\,Byte < Paketlänge \leq 1518\,Byte. \end{cases}$$

In der Formel gibt *Paketrate* die maximale Anzahl der Pakete, die pro Sekunde über eine Netzwerkschnittstelle empfangen oder gesendet werden können, in Abhängigkeit von der Paketlänge an. *Paketlänge* gibt die Länge der Pakete in Byte an. Die Paketlänge beinhaltet den Ethernet Paketkopf, die Nutzdaten und die Prüfsumme. Die minimale Paketlänge bei 10 Gigabit Ethernet beträgt 64 Byte. Enthalten Pakete nicht genug Nutzdaten, um die Mindestlänge zu erreichen, werden sie auf diese Mindestlänge aufgefüllt. Die Maximale Paketlänge beträgt 1518 Byte. Bevor ein Paket über Ethernet versendet wird, wird eine 8 Byte lange Präambel übertragen. Zwischen dem Versenden von zwei Paketen wird eine Pause (das so genannte *Inter-Frame Gap*), die der Dauer einer Übertragung von 12 Byte

entspricht, abgewartet. Durch die Präambel und das *Inter-Frame Gap* ergeben sich die zusätzlichen 20 Byte in der Formel, die auf die Paketlänge addiert werden.

Wie in Abbildung 5.1 dargestellt, können bei Paketlängen von bis zu 64 Byte Paketraten von etwa 14.88 Millionen Paketen pro Sekunde erreicht werden. Mit zunehmender Paketlänge reduziert sich die Paketrate auf bis zu 812744 Pakete pro Sekunde bei der maximalen Paketlänge von 1518 Byte. Untersuchungen wie in [94] zeigen, dass die hohen Paketraten bei geringen Paketlängen eine besondere Herausforderung darstellen. Bei 14.88 Millionen Paketen pro Sekunden bleiben einem Endsystem 67.2 Nanosekunden Zeit für die Verarbeitung eines Paketes. Der Zeitaufwand für Kopieroperationen und Kontextwechsel zwischen Anwendungen und dem Betriebssystem können die Zeit, die für die Verarbeitung eines Pakets zur Verfügung steht, übersteigen. Der Aufwand pro Paket muss also so gering wie möglich gehalten werden, um diese Paketraten zu erreichen.

In existierenden Arbeiten wurde untersucht, welche Paketlängen in verschiedenen Netzen und Anwendungen auftreten: Eine Untersuchung von Datenverkehr im Internet-Backbone [56] zeigt, dass der Großteil der Pakete entweder unter 100 Byte (44% der Pakete) oder über 1400 Byte (37% der Pakete) lang ist. Eine weitere Untersuchung in [104] liefert eine ähnliche Verteilung und zeigt, dass die meisten Pakete entweder 40 Byte (40% der Pakete) oder 1500 Byte (20% der Pakete) lang sind. Eine Untersuchung des Daten-Verkehrs in einem Universitätsnetz [82] bestätigt die Messungen im Internet und zeigt, dass die meisten Pakete unter 100 Byte oder über 1400 Byte lang sind. Auch Untersuchungen des Datenverkehrs in Datenzentren in [10] und [9] zeigen, dass ein großer Anteil des Datenverkehrs aus Paketen besteht, die kleiner als 200 Byte sind. Dabei tauschen die in Datenzentren stark vertretenen Anwendungen MSSQL, HTTP und SMB einen höheren Anteil kleiner Pakete als großer Pakete aus. Darüber hinaus gibt es weitere Beispiele von Anwendungen die kleine Pakete verwenden. Zum Beispiel verwendet Internet-Telefonie vorwiegend kleine Pakete. So hat eine Untersuchung von Voice over IP (VoIP) Verkehr in [103] gezeigt, dass VoIP-Pakete 64 Byte oder 280 Byte lang sind. Auch Online Spiele verwenden vorwiegend kleine Pakete. Eine Untersuchung des Datenverkehrs des Online Spiels Halo 2 [130] und eine Untersuchung des Datenverkehrs von Quake 3 [63] zeigen, dass in Online Spielen Pakete vom Server zu Clients kleiner als 400 Byte und vom Client zum Server kleiner als 200 Byte sind. Ein weiteres Beispiel ist der Anonymisierungsdienst TOR [23]. TOR nutzt 512 Byte große Zellen für die Kommunikation, um Rückschlüsse auf die übertragenen Daten aus den Paketlängen zu verhindern. Es ist daher zu erwarten, dass ein Endsystem in aktuellen Netzen einen hohen Anteil an kleinen Paketen verarbeiten muss. Darüber hinaus ist nicht abzusehen, welche Paketlängen in zukünftigen DsNs verwendet werden. Ein Endsystem sollte daher in der Lage sein, Pakete beliebiger Länge effizient zu verarbeiten. Ein Rahmenwerk, das eine möglichst hohe Leistung bei beliebigen DsNs erreichen soll, muss also auch die maximale Datenrate verfügbarer Netzwerkschnittstellen selbst bei kleinen Paketlängen erreichen können.

5.2 Ursprüngliches NENA Rahmenwerk

In diesem Abschnitt wird näher auf das ursprüngliche NENA Rahmenwerk (NENA) [74] eingegangen. Zunächst werden NENA und seine Anwendungsschnittstelle vorgestellt. Danach wird ein Überblick über den Aufbau eines Endsystems gegeben und der Nachrichtenfluss durch ein Endsystem gezeigt. Anschließend wird näher auf die Nachrichtenverarbeitung in den verschiedenen Komponenten und den Nachrichtenaustausch zwischen den Komponenten eingegangen. Es werden (1) die Nachrichtenverarbeitung innerhalb einer Anwendung, die NENA nutzt, (2) der Nachrichtenaustausch zwischen Anwendung und NENA, (3) die Nachrichtenverarbeitung innerhalb von NENA und (4) der Nachrichtenaustausch zwischen NENA und dem Netz gezeigt. Dabei wird auch auf Leistungsaspekte eingegangen.

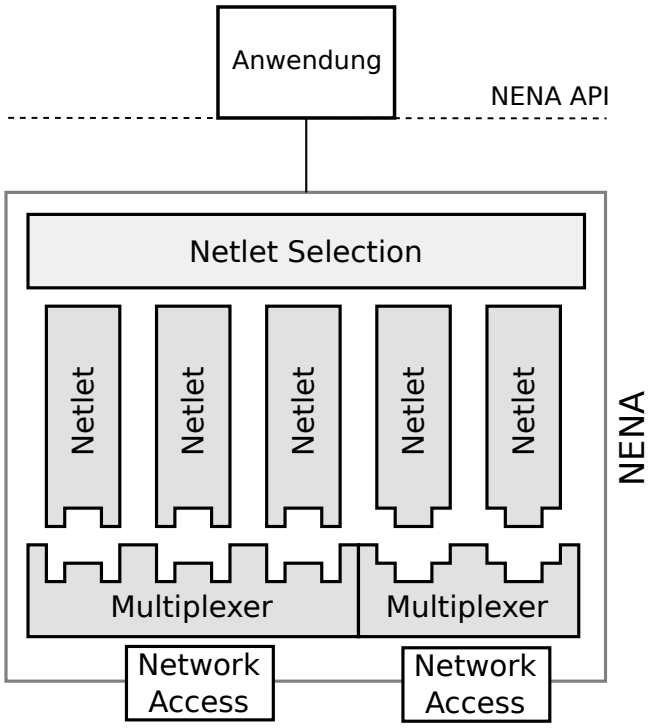

Abbildung 5.2: *Urpsrüngliches NENA Rahmenwerk.*

Das ursprüngliche NENA Rahmenwerk wird in Abbildung 5.2 dargestellt. Es ist als Middleware zwischen Betriebssystem und den Anwendungen, die über NENA kommunizieren, realisiert. Anwendungen verbinden sich mit NENA über eine Anwendungsschnittstelle, die *NENA API* genannt wird. Sie ist eine namensbasierte Schnittstelle. Anwendungen geben in ihren Kommunikationsanfragen den Namen des Dienstes, auf den sie zugreifen wollen, an. Innerhalb von NENA nimmt die *Netlet Selection* die Kommunikationsanfragen der Anwendungen entgegen. Für jede Kommunikationsanfrage wählt

die *Netlet Selection* ein *Netlet* aus. Ein *Netlet* ist ein Container für ein Dienst-spezifisches Protokoll. *Netlets*, die zum gleichen DsN gehören, werden über einen *Multiplexer* gruppiert und an die gemeinsamen *Network Accesses* des DsN angebunden. Ein *Network Access* repräsentiert eine virtuelle Netzwerkschnittstelle, die NENA mit einem DsN verbindet.

5.2.1 NENA API

Die NENA API [71] besteht aus folgenden Primitiven, die von Anwendungen genutzt werden, um Kommunikationsanfragen an NENA zu senden und Kommunikationsbeziehungen zu verwenden.

- *handle = get(name)*
- *handle = put(name)*
- *handle = connect(name)*
- *listen = bind(name)*
- *handle = accept(listen)*
- *data = read(handle)*
- *write(handle, data)*
- *close(handle)*

bind(name) wird benutzt, um einen Dienst über den Namen *name* anzubieten. *accept()* wird benutzt, um Zugriffe auf den über *name* angebotenen Dienst, die über NENA eintreffen, anzunehmen. *get()*, *put()* und *connect()* werden benutzt um auf Dienste zuzugreifen.

get() wird benutzt, um eine Kommunikationsbeziehung mit dem Dienst, der über den Namen *name* identifiziert wird, aufzubauen. Von dieser Kommunikationsbeziehung kann eine Anwendung ausschließlich Daten lesen.

put() wird benutzt, um eine Kommunikationsbeziehung mit dem Dienst, der über den Namen *name* identifiziert wird, aufzubauen. Auf diese Kommunikationsbeziehung kann eine Anwendung ausschließlich Daten schreiben.

connect() wird benutzt, um eine Kommunikationsbeziehung mit dem Dienst, der über den Namen *name* identifiziert wird, aufzubauen. Eine Anwendung kann sowohl von dieser Kommunikationsbeziehung Daten lesen als auch auf diese Kommunikationsbeziehung Daten schreiben.

read() wird benutzt, um Daten von einer Kommunikationsbeziehung zu lesen. *write()* wird benutzt, um Daten auf eine Kommunikationsbeziehung zu schreiben. *close()* wird benutzt, um eine Kommunikationsbeziehung zu beenden.

Die Primitive *get()*, *put()* und *connect()* geben ein so genanntes Handle zurück, das die Kommunikationsbeziehung identifiziert. Das Handle kann verwendet werden, um über Lese- und Schreibbefehle (*read()* und *write()*) Daten von einer Kommunikationsbeziehung zu lesen oder auf eine Kommunikationsbeziehung zu schreiben. Eine Besonderheit sind die Handles, die von *bind()* zurückgegeben und als Parameter von *accept()* verwendet werden. Das von *bind()* zurückgegebene Handle *listen* identifiziert keine Kommunikationsbeziehung wie die anderen Handles und kann daher nicht direkt zum Lesen und Schreiben verwendet werden. Dieses Handle repräsentiert die Bereitschaft der Anwendung, Zugriffe auf den durch *bind()* spezifizierten Namen anzunehmen. Dieses Handle muss als Parameter für *accept()* verwendet werden, um Zugriffe, die von NENA ankommen, zu akzeptieren und damit eine neue Kommunikationsbeziehung aufzubauen. Alle Handles können über den Befehl *close()* wieder geschlossen werden.

5.2.2 Überblick über Endsystem

Ein Endsystem besteht aus Anwendungen, die NENA nutzen, dem NENA Rahmenwerk, dem Betriebssystem des Endsystems und Netzwerkschnittstellen.

In Abbildung 5.3 wird der Aufbau eines Endsystems schematisch dargestellt. NENA ist als Middleware oberhalb des Betriebssystems realisiert. Dadurch wird NENA als regulärer Prozess oberhalb des Betriebssystems ausgeführt. Die Kommunikation über das Netz verläuft über das Betriebssystem. Anwendungen, die NENA für die Netzkommunikation nutzen, werden auch als reguläre Prozesse oberhalb des Betriebssystems ausgeführt. Diese Anwendungen greifen auf NENA zu bzw. kommunizieren mit NENA über Mechanismen des Betriebssystems, so genannte Inter-Prozess-Kommunikation (*Inter Process Communication*, *IPC*). Das Betriebssystem befindet sich unterhalb der Anwendungen und NENA. Innerhalb des Betriebssystems befindet sich der Netzwerkstapel, der gängige Kommunikationsprotokolle wie z.B. IP, TCP und UDP enthält. Das Endsystem ist über Netzwerkschnittstellen (*NIC*) an das Netz angeschlossen. Die Netzwerkschnittstellen werden vom Betriebssystem verwaltet und dazu genutzt, Pakete im Netz zu versenden oder vom Netz zu empfangen.

5.2.3 Nachrichtenfluss durch ein Endsystem

Anwendungen, die NENA für die Netzwerkkommunikation nutzen, tauschen über NENA Nachrichten mit dem Netz aus. Der Datenfluss durch ein Endsystem bei der Kommunikation über NENA wird in Abbildung 5.4 schematisch als Sequenzdiagramm dargestellt. Eine Anwendung kommuniziert über NENA, um z.B. auf einen Dienst im Netz zuzugreifen.

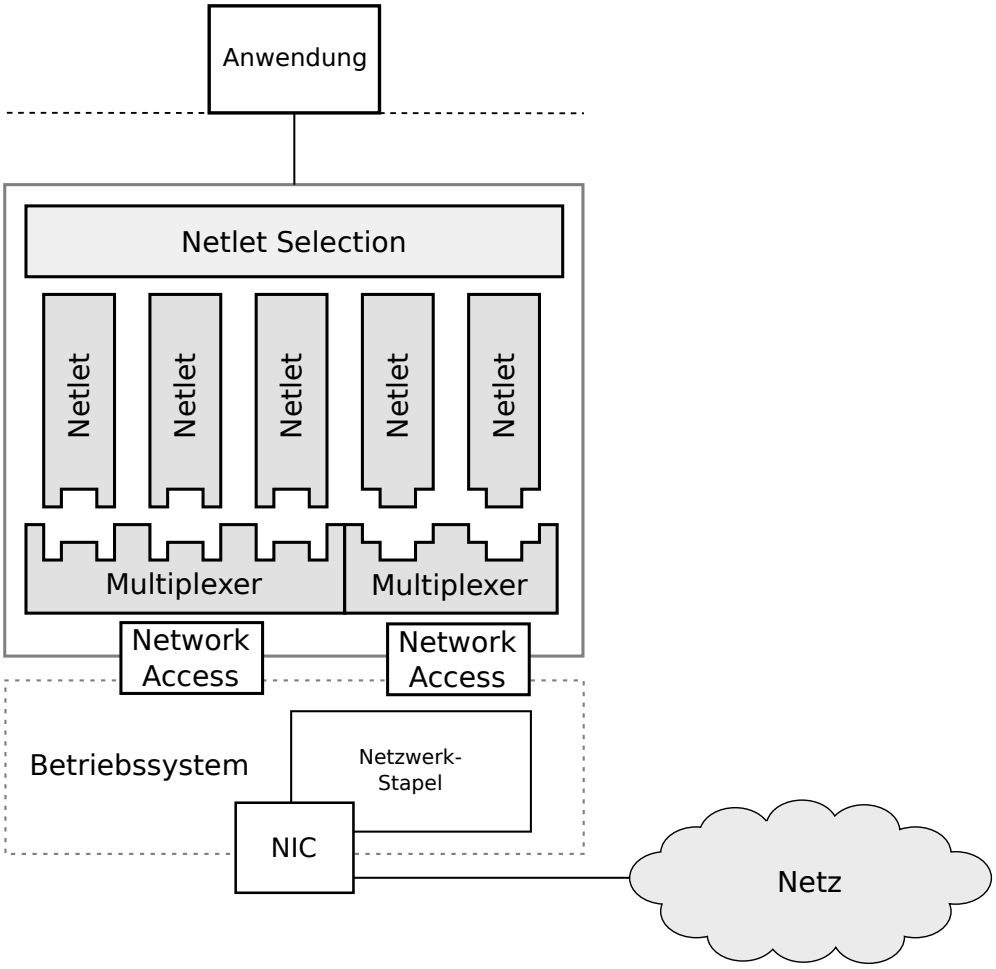

Abbildung 5.3: *Aufbau eines Endsystems*

NENA läuft oberhalb des Betriebssystems des Endsystems. Der Datenfluss wird mit Hilfe von Pfeilen dargestellt.

Nachrichtenfluss von Anwendung in Richtung des Netzes: Wenn die Anwendung auf einen Dienst zugreift, initiiert die Anwendung die Kommunikation (*Request*) mit einer der folgenden Funktionen der NENA-API: *get*, *put*, *connect*. Von der Anwendung wird eine Nachricht an NENA gesendet, die die angeforderte Funktion *get*, *put* oder *connect* und den Namen des angeforderten Dienstes enthält. NENA empfängt die Nachricht und liest die angeforderte Funktion und den angeforderten Namen aus. Anhand dieser beiden Informationen wählt NENA ein für die Kommunikation geeignetes Protokoll. NENA erstellt ein Handle, welches als lokal eindeutiger Identifikator der Kommunikation dient. NENA assoziiert das Handle mit dem ausgewählten Protokoll. Anschließend sendet NENA eine Nachricht, die das Handle enthält, an die Anwendung. Die Anwendung liest das Handle aus der Nachricht aus und verwendet es für die weitere Kommunikation. Ruft

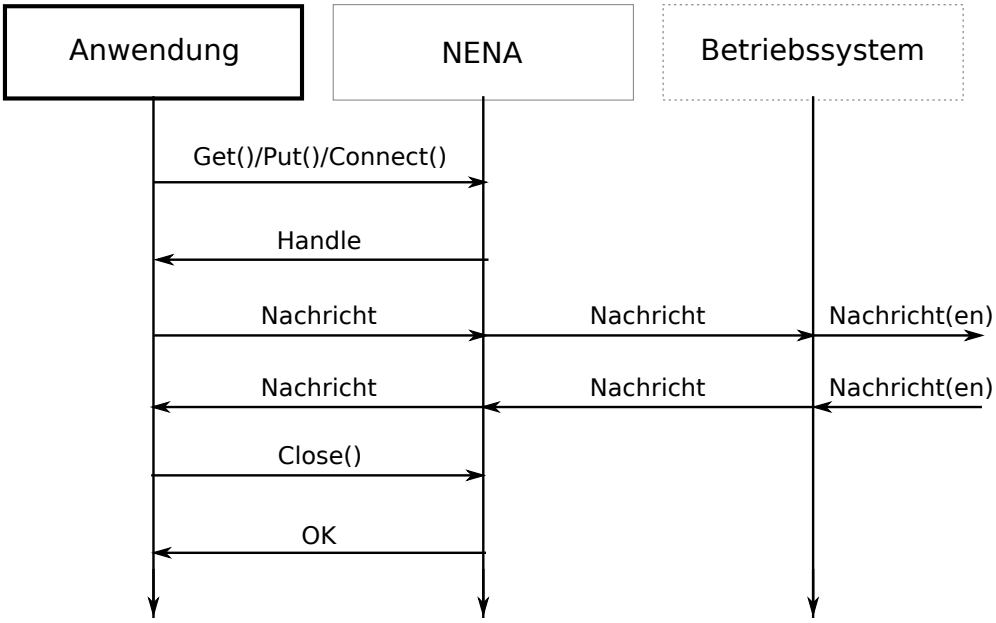

Abbildung 5.4: *Datenfluss bei der Kommunikation über NENA*

die Anwendung einen Schreibbefehl (*write(handle, data)*) auf dem Handle auf, werden die geschriebenen Daten in Nachrichten verpackt. Jede Nachricht enthält das Handle der Kommunikationsbeziehung. Die Nachrichten werden an NENA gesendet. NENA empfängt die Nachrichten, liest das Handle aus und leitet die Nachrichten anhand des Handles weiter an das entsprechende Protokoll in einem Netlet. Das Protokoll verarbeitet die Nachrichten und leitet sie anschließend weiter an den Multiplexer. Der Multiplexer verarbeitet die Nachrichten weiter und versendet sie über einen Network Access. Der Network Access übergibt die Nachrichten an das Betriebssystem. Das Betriebssystem nimmt die Nachrichten entgegen und, je nach Art der virtuellen Verbindung, werden die Nachrichten vom Netzwerkstapel des Betriebssystems weiterverarbeitet (z.B. in UDP und IP Datagramme eingepackt). Schließlich versendet das Betriebssystem die Nachrichten über die Netzwerkschnittstelle des Endsystems.

Nachrichtenfluss vom Netz in Richtung der Anwendungen: Empfängt das Betriebssystem Nachrichten über die Netzwerkschnittstelle des Endsystems, werden die Nachrichten vom Netzwerkstapel weiterverarbeitet. Hierbei werden abhängig von den verwendeten Protokollen Operationen auf den Nachrichten ausgeführt, z.B. UDP- und IP-Paketköpfe entfernt. Nach der Verarbeitung im Netzwerkstapel leitet das Betriebssystem die Nachrichten weiter an NENA. NENA nimmt die Nachrichten über den Network Access vom Betriebssystem entgegen. Der Multiplexer verarbeitet die Nachrichten und leitet sie weiter an das Protokoll in einem Netlet, welches die Nachrichten weiterverarbeitet. Die Nachrichten werden mit dem Handle der Kommunikationsbeziehung versehen und an die

Anwendung weitergereicht. Die Anwendung nimmt bei einem Aufruf eines Lesebefehls (*data = read(handle)*) die Nachrichten, die zur Kommunikationsbeziehung gehören, von NENA entgegen. Schließlich kann die Anwendung die in den Nachrichten enthaltenen Daten auslesen und weiterverarbeiten.

Beenden der Kommunikationsbeziehung: Wenn die Anwendung nicht mehr auf einen Dienst zugreift, kann sie die Kommunikationsbeziehung mit der Funktion *close(handle)*, die das Handle der Kommunikationsbeziehung spezifiziert, beenden. Daraufhin wird eine Nachricht mit dem Befehl *close* an NENA gesendet. NENA nimmt die Nachricht entgegen, beendet die Kommunikationsbeziehung, und bestätigt das Beenden der Kommunikationsbeziehung mit der Nachricht *OK*.

Im Folgenden wird näher auf die Nachrichtenverarbeitung im Endsystem eingegangen. Es werden die Nachrichtenverarbeitung in Anwendungen, der Nachrichtenaustausch zwischen Anwendungen und NENA, die Nachrichtenverarbeitung innerhalb von NENA selbst und der Nachrichtenaustausch zwischen NENA und Netz diskutiert.

5.2.4 Nachrichtenverarbeitung in Anwendungen

Anwendungen bestehen aus zwei Komponenten: der Anwendungslogik und dem NENA-API Code. Die Anwendungslogik enthält die Funktionalität der Anwendung. Der NENA-API Code enthält die Funktionalität für den Nachrichtenaustausch mit NENA. Der Aufbau von Anwendungen und die Nachrichtenverarbeitung in Anwendungen wird in Abbildung 5.5 schematisch dargestellt.

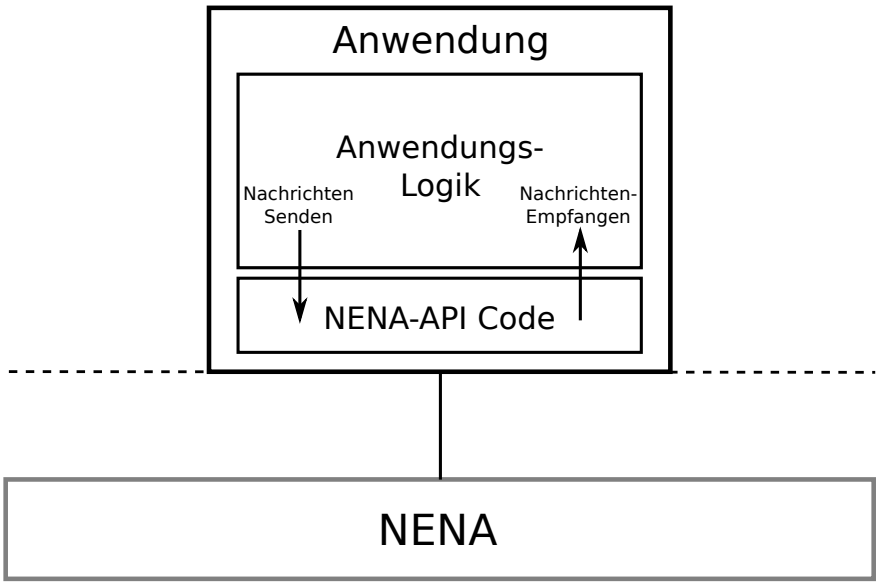

Abbildung 5.5: *Nachrichtenverarbeitung in Anwendung.*

Zum Versenden und Empfangen von Nachrichten über NENA verwenden Anwendungen die NENA-API, die im NENA-API Code realisiert wird. Eine Anwendung erstellt Nachrichten in der Anwendungslogik und versendet sie mit Hilfe des NENA-API Codes. Die Anwendung empfängt Nachrichten von NENA mit Hilfe des NENA-API Codes und verarbeitet sie in der Anwendungslogik weiter. Nach welchem Muster die Anwendung Nachrichten erstellt und versendet, hängt von der Anwendung und von den verwendeten Diensten ab. Nach welchem Muster die Anwendung Nachrichten empfängt, hängt ebenfalls von der Anwendung und von den verwendeten Diensten ab.

Damit eine Anwendung auf einem Endsystem die maximale Datenrate der Netzwerkschnittstellen des Endsystems erreichen kann, muss die Anwendung mindestens die Datenrate generieren (gesendete Daten) können bzw. verarbeiten (empfangene Daten) können, die die Netzwerkschnittstellen des Endsystems erreichen können. Verwenden mehrere Anwendungen NENA, können mehrere Anwendungen zusammen die maximale Datenrate der Netzwerkschnittstellen erreichen, auch wenn einzelnen Anwendungen dies nicht möglich ist. Im Rahmen dieser Arbeit wird allerdings, sofern nicht anders dargelegt, davon ausgegangen, dass eine Anwendung mindestens die Datenrate generieren und verarbeiten kann, die benötigt wird, um die Netzwerkschnittstellen des Endsystems auszulasten.

Da die Anwendungen von Dritten erstellt werden, müssen Leistungsoptimierungen von eben diesen Dritten selbst umgesetzt werden. NENA hat keinen Einfluss auf die Implementierung bzw. die Leistungsfähigkeit der Anwendungen. Hier können lediglich Empfehlungen ausgesprochen werden, wie man die Anwendungen auf Leistung optimieren kann, wie z.B. die Verwendung mehrerer Threads oder die Optimierungen für Cache-Zugriffe.

5.2.5 Nachrichtenaustausch zwischen Anwendung und NENA

Anwendungen und NENA selbst werden als reguläre Prozesse oberhalb des Betriebssystems ausgeführt. Daher setzten Anwendungen und NENA IPC-Mechanismen des Betriebssystems ein, um miteinander zu interagieren.

Der Nachrichtenaustausch zwischen einer Anwendung und NENA wird in Abbildung 5.6 schematisch dargestellt. Eine Anwendung übergibt Nachrichten zum Versand an NENA über die NENA-API. Hierzu verwendet sie den NENA-API Code. Beim Empfang von Nachrichten nimmt eine Anwendung die Nachrichten von NENA über die NENA-API entgegen. Auch hierfür wird wieder der NENA-API Codes verwendet. Versendet die Anwendung Nachrichten an NENA, werden diese dem Betriebssystem übergeben. Das Betriebssystem reicht die Nachrichten weiter an NENA, das die Nachrichten vom Betriebssystem entgegennimmt. Sendet NENA Nachrichten an die Anwendungen, verläuft der Nachrichtenaustausch analog. NENA übergibt die Nachrichten dem Betriebssystem,

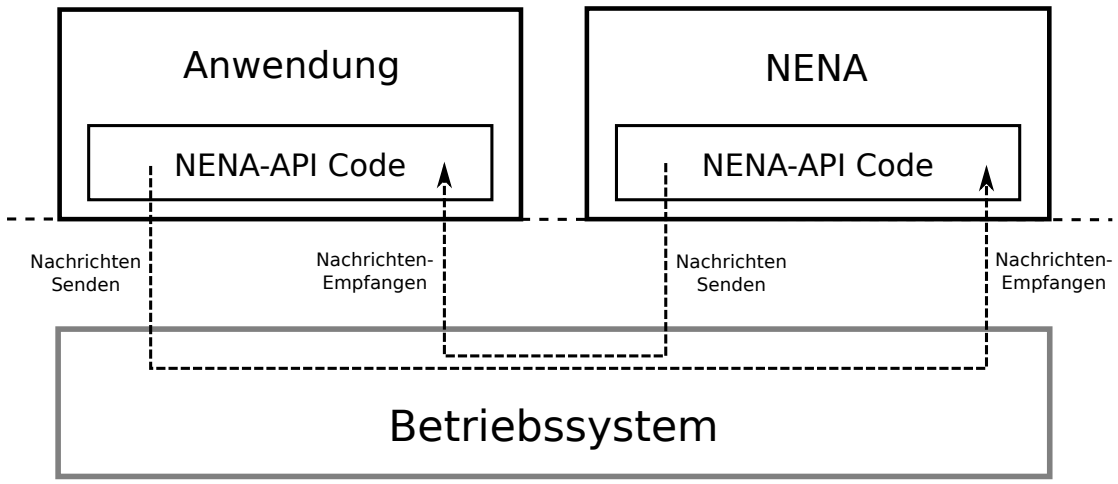

Abbildung 5.6: *Nachrichtenaustausch zwischen Anwendung und NENA.*

welches die Nachrichten an die Anwendung weiterreicht. Die Anwendung nimmt im NENA-API Code die Nachrichten entgegen und verarbeitet sie weiter.

Damit die maximale Leistung der Netzwerkschnittstellen des Endsystems in einer Anwendung erreicht werden kann, muss der Nachrichtenaustausch zwischen Anwendungen und NENA mindestens die maximale Leistung der Netzwerkschnittstellen erreichen. Die verwendeten Mechanismen zum Nachrichtenaustausch müssen also diese Leistung erbringen können.

5.2.6 Nachrichtenverarbeitung in NENA

Erhält NENA von Anwendungen oder aus dem Netz Nachrichten werden diese von Protokollen, Multiplexern und Network Accesses weiterverarbeitet. NENA erhält die Nachrichten über das Betriebssystem. Falls eine Anwendung die Quelle ist, erhält NENA die Nachrichten über IPC-Mechanismen. Falls die Nachrichten vom Netz kommen, erhält NENA die Nachrichten über die in den Network Accesses realisierten Mechanismen.

Der Nachrichtenfluss bei der Nachrichtenverarbeitung in NENA wird vereinfacht in Abbildung 5.7 dargestellt. Am oberen Ende nimmt NENA Nachrichten von Anwendungen zum Versand über das Netz entgegen. Diese Nachrichten werden vom Betriebssystem über IPC-Mechanismen, die im NENA-API Code implementiert sind, erhalten. NENA ordnet jede Nachricht einem Netlet bzw. Protokoll zu, das innerhalb von NENA geladen ist. Die Nachricht wird von dem Protokoll weiterverarbeitet und an den Multiplexer, an den das Protokoll gebunden ist, weitergereicht. Der Multiplexer verarbeitet die Nachricht weiter und übergibt sie einem Network Access. Der Network Access versendet schließlich die Nachricht mit Hilfe des Betriebssystems über das Netz. Am unteren Ende nimmt NENA Nachrichten vom Netz mit Hilfe des Betriebssystems entgegen. Diese Nachrichten

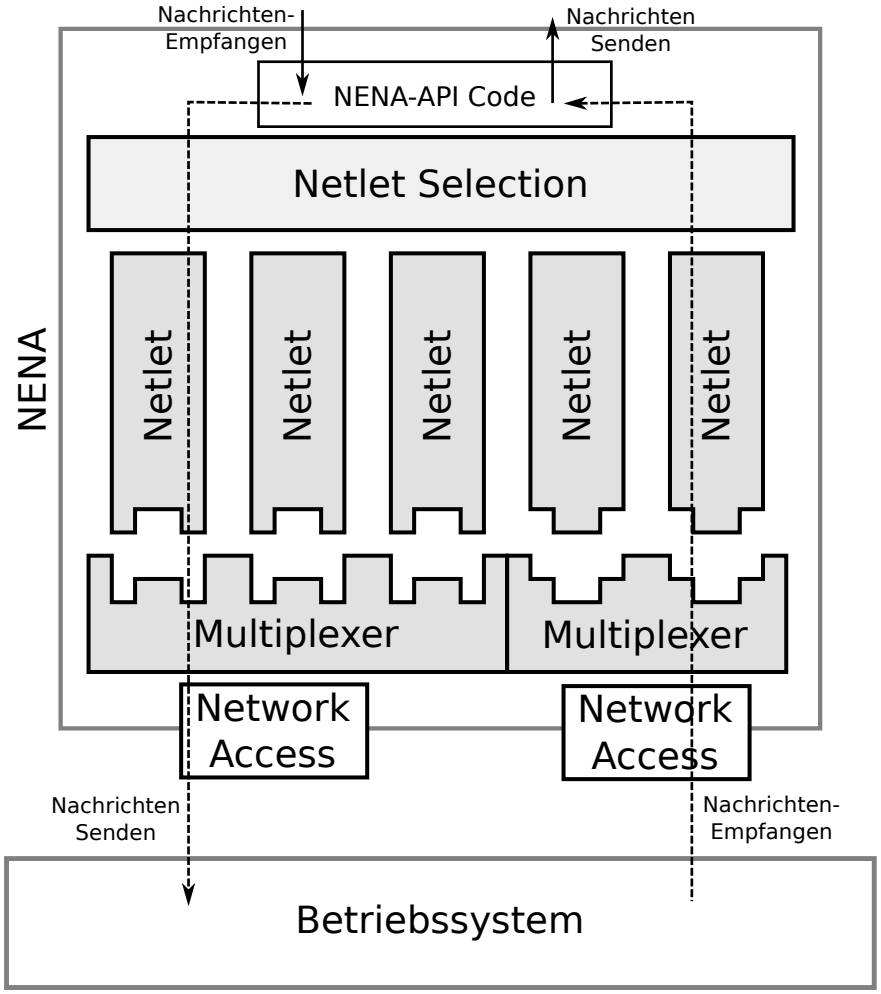

Abbildung 5.7: *Nachrichtenverarbeitung in NENA.*

werden über Network Accesses empfangen. Jede dieser Nachrichten wird von einem Multiplexer weiterverarbeitet. Dieser wählt ein Netlet und reicht die Nachricht an dieses weiter. Das Protokoll verarbeitet die Nachricht weiter. Schließlich wird die Nachricht einer Anwendung zugeordnet und über den NENA-API Code an diese Anwendung weitergereicht. Der NENA-API Code verwendet IPC-Mechanismen des Betriebssystems, um die Nachricht der Anwendung zu übergeben.

Damit die Leistung der Netzwerkschnittstellen des Endsystems erreicht werden kann, muss die Nachrichtenverarbeitung innerhalb von NENA mindestens die maximale Leistung der Netzwerkschnittstellen erreichen. Beim Versand von Nachrichten von einer Anwendung bedeutet dies, dass der Empfang von Nachrichten im NENA-API Code, die Zuordnung von Nachrichten zu Protokollen, die Verarbeitung der Nachrichten in Protokollen und Multiplexern und der Versand über den Network Access schnell genug sein müssen. Beim Empfang von Nachrichten bedeutet es, dass der Empfang von Nachrichten

im Network Access, die Verarbeitung im Multiplexer, die Zuordnung zu einem Protokoll, die Verarbeitung im Protokoll, die Zuordnung zu einer Anwendung und der Versand über den NENA-API Code schnell genug sein müssen.

Da Protokolle und Multiplexer von Dritten erstellt werden, hat NENA keinen Einfluss auf die Verarbeitung der Nachrichten innerhalb dieser Komponenten. Somit ist die Optimierung der Leistung der Protokolle und Multiplexer selbst Aufgabe derjenigen, die sie erstellen. Wie bei Anwendungen können hier lediglich Empfehlungen ausgesprochen werden, um die Leistung zu optimieren. Die übrigen Schritte der Nachrichtenverarbeitung in NENA können allerdings optimiert werden.

5.2.7 Nachrichtenaustausch zwischen NENA und Netz

Versendet oder empfängt NENA Nachrichten über das Netz, benutzt NENA verschiedene Network Accesses. Die Network Accesses nutzen wiederum Mechanismen des Betriebssystems für den Nachrichtenaustausch über die Netzwerkschnittstellen des Endsystems.

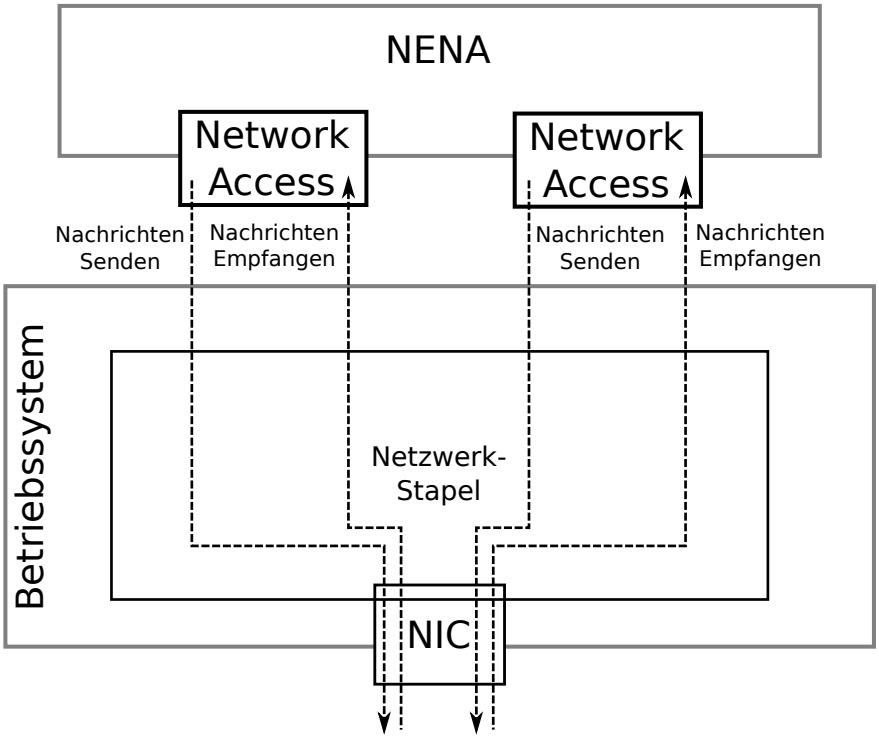

Abbildung 5.8: *Nachrichtenaustausch zwischen NENA und Netz.*

Der Nachrichtenaustausch zwischen NENA und Netz wird in Abbildung 5.8 dargestellt. NENA übergibt dem Betriebssystem Nachrichten zum Versand über die Netzwerkschnittstellen des Endsystems. Das Betriebssystem nimmt die Nachrichten entgegen und verarbeitet sie im Netzwerkstapel weiter. Hierbei können die Nachrichten beispielsweise

in UDP und IP Paketen verpackt werden. Schließlich versendet das Betriebssystem die fertigen Pakete über eine Netzwerkschnittstelle des Endsystems. Das Betriebssystem empfängt Pakete vom Netz über die Netzwerkschnittstellen. Beim Empfang von Paketen über eine Netzwerkschnittstelle verarbeitet das Betriebssystem die Pakete im Netzwerkstapel. Dabei entpackt das Betriebssystem die in den Paketen enthaltenen Nachrichten und reicht sie an NENA weiter. NENA nimmt die Nachrichten vom Betriebssystem entgegen und verarbeitet sie schließlich weiter.

Damit die Leistung der Netzwerkschnittstellen des Endsystems erreicht werden kann, muss der Nachrichtenaustausch zwischen NENA und Netz mindestens die maximale Leistung der Netzwerkschnittstellen erreichen.

5.2.8 Zusammenfassung

Um eine möglichst hohe Leistung bei der Kommunikation von Anwendungen über das Netz mit Hilfe des NENA Rahmenwerkes zu erreichen, müssen verschiedene Schritte der Nachrichtenverarbeitung optimiert werden. Das beinhaltet (1) die Datenverarbeitung in Anwendungen, (2) den Datenaustausch zwischen Anwendungen und NENA, (3) die Datenverarbeitung innerhalb von NENA und (4) den Datenaustausch zwischen NENA und Netz. Allerdings hat NENA nicht Einfluss auf die Leistungsfähigkeit all dieser Schritte. Anwendungen, Protokolle und Multiplexer werden von Dritten entwickelt. Daher muss ihre Leistung von denjenigen optimiert werden, die sie erstellen. Somit hat NENA nur Einfluss auf die Leistungsfähigkeit der folgenden Schritte:

- Datenaustausch zwischen Anwendungen und NENA
- Datenverarbeitung innerhalb von NENA
- Datenaustausch zwischen NENA und Netz

Außerdem wird der Fokus der Leistungsverbesserungen auf den Austausch von Daten-Nachrichten gelegt. Kommuniziert eine Anwendung über NENA, werden vor dem eigentlichen Datenaustausch Kontroll-Nachrichten ausgetauscht. Diese Kontroll-Nachrichten initiieren zum Beispiel Kommunikationsbeziehungen oder Virtuelle Verbindungen zu DsNs. Damit erzeugen die Kontroll-Nachrichten auch zusätzlichen Aufwand in NENA, zum Beispiel bei der Auswahl von Protokollen. Abhängig von der Dauer einer Kommunikationsbeziehung werden mehr Daten- als Kontroll-Nachrichten oder mehr Kontroll- als Daten-Nachrichten ausgetauscht. Im Rahmen dieser Arbeit wird davon ausgegangen, dass bei einem Großteil der Kommunikationsbeziehungen mehr Daten- als Kontroll-Nachrichten ausgetauscht werden. Deswegen wird die Optimierung des Austauschs von Kontroll-Nachrichten vernachlässigt.

5.3 Möglichkeiten für schnellen Datenaustausch zwischen Anwendungen und Netz

In diesem Abschnitt werden Möglichkeiten für einen schnellen Datenaustausch zwischen Anwendungen und dem Netz über ein Rahmenwerk für DsNs vorgestellt. Zunächst wird in Abschnitt 5.3.1 der Datenaustausch zwischen dem Rahmenwerk und dem Netz über das Betriebssystem betrachtet. Durch Optimierungen, wie das Vermeiden von Kontext-Wechseln zwischen Anwendungen und dem Betriebssystem und durch das Vermeiden des Kopieren der Daten, sind Leistungssteigerungen möglich. In Abschnitt 5.3.2 wird der Datenaustausch zwischen dem Rahmenwerk und den Anwendungen betrachtet. Es werden die Mechanismen heutiger Betriebssysteme *PIPE/FIFO*, *Socket* und *Shared Memory* vorgestellt und miteinander hinsichtlich ihrer Leistung verglichen. Durch das Vermeiden von Kopien und Kontext-Wechseln, sowie Burst-Verarbeitung, kann *Shared Memory* die höchste Leistung erreichen. In Abschnitt 5.3.3 wird die Datenverarbeitung innerhalb des Rahmenwerkes betrachtet. Durch das Vermeiden von Kopieren der Daten und die Parallelisierung der Datenverarbeitung kann die Leistung gesteigert werden. Bei der Parallelisierung sollte zudem auf Eigenschaften des Endsystems geachtet werden, um z.B. bei der Verwendung von *Hyper-Threading* oder durch *NUMA*-Effekte die Leistung nicht zu reduzieren.

5.3.1 Datenaustausch zwischen Rahmenwerk und Netz

In diesem Abschnitt wird der Datenaustausch zwischen dem Rahmenwerk und dem Netz über die Netzwerkschnittstellen des Endsystems behandelt. Als Grundlage wird der Datenaustausch ohne Leistungsoptimierungen über das Betriebssystem vorgestellt. Basierend hierauf werden die Hindernisse für einen schnellen Datenaustausch diskutiert und anschließend verschiedene Leistungsoptimierungen vorgestellt.

5.3.1.1 Datenaustausch über das Betriebssystem

Als Grundlage für die Diskussion der Hindernisse und der verschiedenen Optimierungen für den schnellen Datenaustausch zwischen dem Rahmenwerk und dem Netz über die Netzwerkschnittstellen des Endsystems dient der Datenaustausch von Anwendungen über das Betriebssystem.

Der Datenaustausch über das Betriebssystem ist schematisch in Abbildung 5.9 dargestellt. Eine Anwendung auf einem Endsystem (genauer: Prozess) wie z.B. ein Rahmenwerk für DsNs, die über das Netzwerk kommuniziert, wird oberhalb des Betriebssystems ausgeführt. Anwendungen werden im so genannten User-Mode ausgeführt. Der User-Mode ist ein unprivilegierter Ausführungsmodus, in dem nur ein eingeschränkter Zugriff

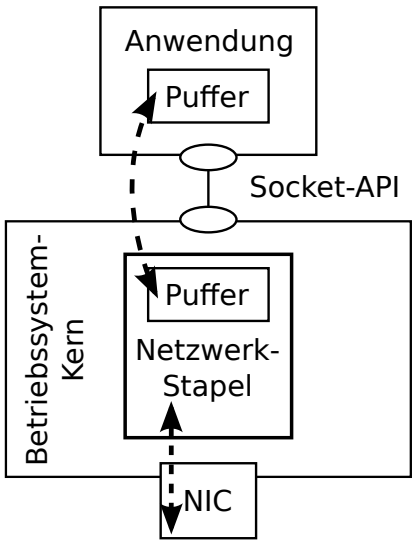

Abbildung 5.9: *Verwendung der Socket-API zur Kommunikation.*

auf den Speicher, auf die Prozessor-Befehle und auf die Hardware eines Endsystems möglich ist. Unterhalb der Anwendung sitzt das Betriebssystem bzw. der Betriebssystem-Kern. Der Betriebssystem-Kern wird im so genannten Kernel-Mode ausgeführt. Der Kernel-Mode ist ein privilegierter Ausführungsmodus, in dem der volle Zugriff auf den Speicher, auf Prozessor-Befehle und Hardware eines Endsystems möglich ist. Der Betriebssystem-Kern verwaltet somit den Zugriff auf die Netzwerkschnittstellen des Endsystems mit Hilfe der Treiber der Netzwerkschnittstellen. Außerdem verwaltet der Betriebssystem-Kern die Ausführung von Anwendungen und beinhaltet den Netzwerkstapel, der gängige Kommunikationsprotokolle wie z.B. IP, TCP und UDP enthält.

Damit eine Anwendung Daten über das Netzwerk versenden oder vom Netzwerk empfangen kann, verwendet sie die Socket-API, eine Schnittstelle des Betriebssystems. Die Socket-API enthält Funktionen zum Versenden und Empfangen von Daten. Beim Versenden von Daten über die Sende-Funktion übergibt die Anwendung einen Puffer in ihrem eigenen Speicher mit den zu sendenden Daten. Beim Empfangen von Daten über die Empfangsfunktion übergibt die Anwendung einen Puffer in ihrem eigenen Speicher für die zu empfangenden Daten. Außerdem wird von der Anwendung spezifiziert, welche Protokolle und Adressen verwendet werden sollen. Anschließend ruft die Socket-API einen so genannten Systemaufruf auf, der dafür sorgt, dass die Ausführung von der Anwendung in den Betriebssystem-Kern wechselt. Beim Versenden von Daten kopiert der Betriebssystem-Kern die Daten aus dem Puffer der Anwendung in den Speicherbereich des Betriebssystems. Daraufhin verarbeitet der Betriebssystem-Kern die Daten im Netzwerkstapel des Betriebssystem. Dabei werden die Daten von den von der Anwendung spezifizierten Protokollen weiterverarbeitet und z.B. Paketköpfe hinzugefügt oder die Daten auf mehrere kleinere Dateneinheiten aufgeteilt. Nach der Verarbeitung im Netz-

werkstapel werden die Daten über den Treiber der Netzwerkschnittstelle versendet. Der Treiber sorgt dafür, dass die Daten über die Netzwerkschnittstelle übertragen werden. Außerdem informiert der Betriebssystem-Kern die Anwendung über den Versand der Daten und setzt die Ausführung der Anwendung fort[1].

Beim Empfang von Daten wird der Treiber von der Netzwerkschnittstelle über eine so genannte Unterbrechung darüber informiert, dass Daten empfangen worden sind. Daraufhin werden die von der Netzwerkschnittstelle empfangenen Daten durch den Netzwerkstapel des Betriebssystems verarbeitet. Dabei werden die verwendeten Protokolle ermittelt und die Daten durch die entsprechenden Protokolle verarbeitet. Dabei werden z.B. Paketköpfe entfernt oder mehrere erhaltene Dateneinheiten gepuffert und zu zusammenhängenden Daten zusammengesetzt. Nach der Verarbeitung der Daten durch den Netzwerkstapel werden die Daten für den Abruf durch die entsprechenden Anwendungen bereit gehalten. Ruft eine Anwendung die Empfangsfunktion auf, ruft die Socket-API einen Systemaufruf auf, der dafür sorgt, dass die Ausführung von der Anwendung in den Betriebssystem-Kern wechselt. Durch diesen Aufruf wird der Betriebssystem-Kern über die Bereitschaft der Anwendung informiert, Daten über das Netz zu empfangen. Der Betriebssystem-Kern kopiert nun die bereit gehaltenen Daten aus dem Speicher des Betriebssystems in den Puffer der Anwendung im Speicherbereich der Anwendung und setzt die Ausführung der Anwendung fort. Abschließend verarbeitet die Anwendung die Daten in ihrem eigenen Puffer gemäß der Anwendungslogik weiter.

Leistung

Für die Betrachtung der Leistung beim Datenaustausch über das Betriebssystem ohne Optimierungen wurden die möglichen Paketraten bei variierenden Paketlängen anhand eines Experiments gemessen. Bei diesem wurden die Paketraten zwischen zwei Endsystemen, die direkt über eine 10 Gigabit/s Verbindung miteinander verbunden sind, bei unterschiedlichen Paketlängen gemessen. Beide Endsysteme verwenden einen Intel i7-4770 Prozessor mit vier Prozessor-Kernen, 16 GB RAM und eine Intel x520-DA2 Netzwerkkarte. Das Betriebssystem beider Endsysteme ist Ubuntu Linux 14.04 mit dem Linux Kernel der Version 3.13.0. Auf einem Endsystem wird eine Anwendung ausgeführt, die so schnell wie möglich Pakete an das andere Endsystem versendet. Auf dem anderen empfängt eine Anwendung so schnell wie möglich die Pakete und misst dabei die Paketrate. Beide Anwendungen verwenden für den Datenaustausch die Socket-API des Betriebssystems (genauer: die Socket-Variante *AF_PACKET Sockets*). Die übertragenen Pakete bestehen aus einem Ethernet-, einem IP-, und einem UDP-Paketkopf. Zusätzlich werden sie auf die untersuchte Paketlänge mit Hilfe von Nutzdaten aufgefüllt.

[1] Anmerkung: je nach Aufbau des Endsystems und der Anwendung kann die Ausführung der Anwendung bereits nach dem Kopieren der Daten aus dem Speicher der Anwendung in den Speicher des Betriebssystems fortgesetzt werden.

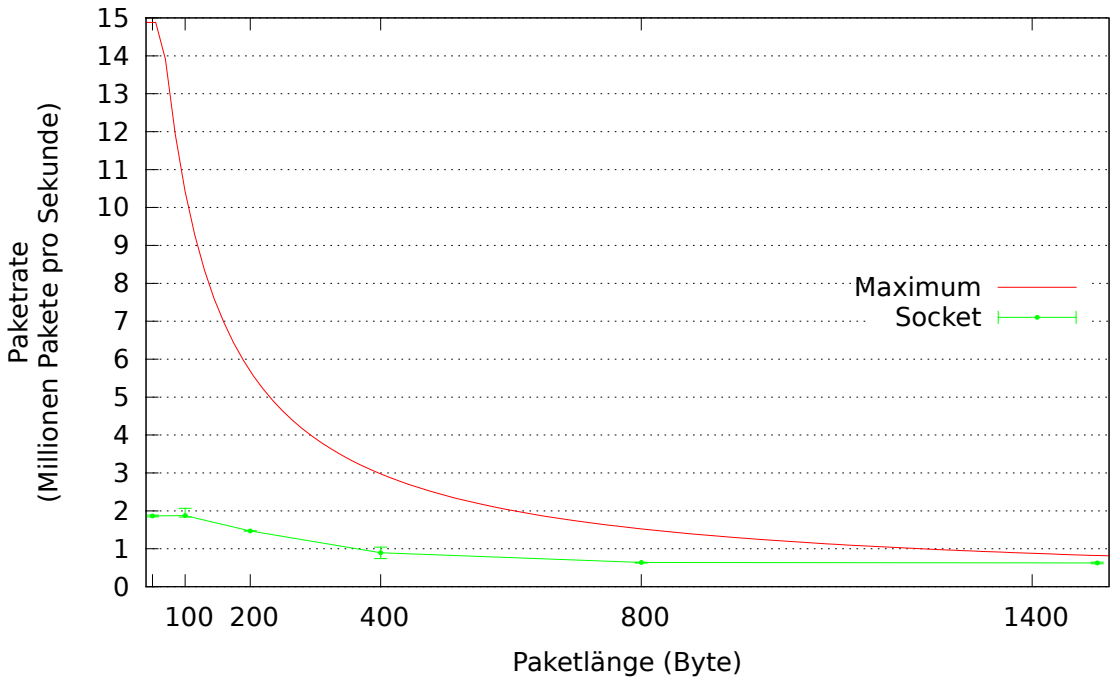

Abbildung 5.10: *Paketraten beim Datenaustausch über das Betriebssystem mit variierenden Paketlängen.*

In Abbildung 5.10 werden die Ergebnisse der Leistungsmessungen dargestellt. Auf der x-Achse werden die Paketlängen in Byte dargestellt. Es wurden Paketlängen von 50, 100, 200, 400, 800 und 1500 Byte untersucht. Auf der y-Achse werden die gemessenen Paketraten in Millionen Paketen pro Sekunde (MPPS) dargestellt. Für jede Paketlänge wurden zehn Versuchsläufe durchgeführt, in denen jeweils die Zeit zur Übertragung von 100 Millionen Paketen gemessen und die Paketrate ermittelt wurde. Die Kurve *Socket* zeigt diese Paketraten, wobei für jede Paketlänge der Durchschnittswert (arithmetisches Mittel), das Minimum und das Maximum über die zehn Versuchsläufe mit Hilfe von Fehlerbalken dargestellt werden. Für einen Vergleich zeigt die Kurve *Maximum* die maximal mögliche Paketrate der 10 Gigabit/s Netzwerkschnittstelle. Wie in der Abbildung zu sehen, erreicht der Datenaustausch über das Betriebssystem mit der Socket-API nicht die mögliche Leistung der Netzwerkschnittstelle. Es wird eine maximale Paketrate von etwa 2.1 MPPS erreicht. Im Durchschnitt bleiben die Paketraten unter 1.9 MPPS. Beide Werte liegen deutlich unter den möglichen 14.88 MPPS der 10 Gigabit/s Netzwerkschnittstelle.

5.3.1.2 Hindernisse für schnellen Datenaustausch

Wie im vorherigen Abschnitt gezeigt, erreicht der Datenaustausch über das Betriebssystem mit Hilfe der Socket-API nicht die Daten- und Paketraten, die mit 10 Gigabit Ethernet

Schnittstellen möglich sind. Daher werden anhand des zuvor gezeigten Datenaustauschs über das Betriebssystem in diesem Abschnitt Probleme und Herausforderungen in heutigen Betriebssystemen für die schnelle Paketverarbeitung präsentiert. Mögliche Hindernisse für die schnelle Paketverarbeitung sind Speicher-Kopien, Systemaufrufe bzw. der Kontext-Wechsel zwischen Betriebssystem und Anwendung, Interrupt-getriebene Netzwerk-Treiber, sowie Effekte durch Speicherzugriffe und den Aufbau des Prozessors bzw. Endsystems.

Wie in Abschnitt 5.1 gezeigt, kann bei minimaler Paketgröße eine 10 Gigabit/s Netzwerkschnittstelle 14.88 Millionen Pakete pro Sekunde übertragen. Pro Paket bleiben einem Endsystem damit 67.2 Nanosekunden Zeit für dessen Verarbeitung. Der Aufwand pro Paket muss also so gering wie möglich gehalten werden, um diese Paketraten zu erreichen.

In [94] wurde eine Analyse der Socket-API durchgeführt. Die Socket-Schnittstelle, so wie sie im Datenaustausch über das Betriebssystem verwendet wird, kopiert in einem Systemaufruf jedes Paket, das gesendet wird, vom Speicher der Anwendung in den Speicher des Betriebssystems. Genauso kopiert sie jedes Paket, das empfangen wird, in einem Systemaufruf vom Speicher des Betriebssystems in den Speicher der Anwendung. Diese Kopieroperationen nehmen viel Zeit in Anspruch. Darüber hinaus benötigt der Systemaufruf, bei dem die Ausführung vom Kontext der Anwendung in den Kontext des Betriebssystems wechselt, selbst Zeit. Der Zeitaufwand eines Systemaufrufs übersteigt sogar die Zeit, die für die Verarbeitung eines Pakets zur Verfügung steht. Beim Empfang von Paketen vom Netz zeigt sich ein weiteres Problem. Werden Interrupt-basierte Netzwerk-Treiber verwendet, wird beim Empfang jedes Pakets eine Unterbrechungsroutine des Betriebssystems ausgeführt, in der das Paket weiterverarbeitet wird. Bei sehr hohen Paketraten wird diese Routine entsprechend oft aufgerufen und dementsprechend hoch ist der Zeitaufwand.

Ein weiteres Problem können Speicherzugriffe sein. Wird auf Daten eines Pakets zugegriffen und diese Daten befinden sich nicht im schnellen Cache-Speicher des Prozessors, müssen die Daten zunächst aus höheren Cache-Ebenen oder aus dem Arbeitsspeicher des Endsystems ausgelesen werden. Die Dauer für den Zugriff auf Daten aus dem Arbeitsspeicher kann einen großen Teil der für die Bearbeitung eines Pakets zur Verfügung stehenden Zeit benötigen.

Ein ähnliches Problem stellen Seiteneffekte durch andere Prozesse auf dem Endsystem dar. Wird eine Anwendung, die Pakete verarbeitet, durch eine andere Anwendung vom Prozessor verdrängt, muss die Paket-verarbeitende Anwendung nicht nur eine Zeit lang warten, bis sie wieder auf dem Prozessor ausgeführt wird. Darüber hinaus kann die Abarbeitung anderer Prozesse auf dem Prozessor für die Paketverarbeitung relevante Daten aus dem Cache des Prozessors verdrängen. Hierdurch müssen die entsprechenden Daten bei der erneuten Ausführung der Paket-verarbeitenden Anwendung erneut aus dem Speicher ausgelesen werden.

Außerdem kann der Aufbau des Prozessors bzw. der Computer-Architektur des Endsystems einen Einfluss auf die Leistung einer Paket-verarbeitenden Anwendung haben. Verwendet der Prozessor zum Beispiel eine Technologie wie *HyperThreading*, um mehrere logische Prozessor-Kerne auf einem physikalischen Prozessor-Kern zu betreiben, kann die Ausführung einer Paket-verarbeitenden Anwendung auf einem logischen Prozessor-Kern durch die Ausführung einer anderen Anwendung auf einem anderen logischen Prozessor-Kern beeinträchtigt werden. Verwendet ein Endsystem mehrere Prozessoren, kann es außerdem zu so genannten *NUMA*-Effekten kommen. In Mehr-Prozessor-Systemen können Teile des Arbeitsspeichers jeweils an unterschiedliche Prozessoren angebunden sein. Der Zugriff eines Prozessors auf den Speicher, der an einen anderen Prozessor angeschlossen ist, verläuft über eine vergleichsweise langsame Verbindung zwischen den Prozessoren. Hierdurch ist der Zugriff eines Prozessors auf den Arbeitsspeicher unterschiedlich schnell. Liegen also für die Paketverarbeitung relevante Daten im Speicher, der an einen anderen Prozessor angeschlossen ist als auf welchem die Paket-verarbeitende Anwendung ausgeführt wird, kann dies die Leistung reduzieren.

5.3.1.3 Optimierungen für schnellen Datenaustausch

Um eine möglichst hohe Leistung bei der Paketverarbeitung zu erhalten, müssen die im vorherigen Abschnitt präsentierten Hindernisse für einen schnellen Datenaustausch umgangen werden. Hierzu werden fünf Ansätze vorgestellt.

(1) Ein Ansatz ist das Vermeiden von Speicher-Kopien. Bei so genannten Zero-Copy Ansätzen wird versucht, den Datenaustausch zwischen Anwendung und Netz ohne Speicher-Kopien durchzuführen. Hierzu wird ein gemeinsamer Speicher-Bereich verwendet, den sich die Anwendung und die Netzwerkschnittstelle teilen. Beim Senden schreibt die Anwendung Pakete direkt in den Speicher-Bereich und die Netzwerkschnittstelle versendet sie aus dem Speicher-Bereich. Beim Empfang schreibt die Netzwerkschnittstelle Pakete in diesen Speicher-Bereich und die Anwendung liest sie direkt aus dem Speicher.

(2) Ein zweiter Ansatz ist das Vermeiden von Kontext-Wechseln zwischen dem Betriebssystem und den Anwendungen. Eine Möglichkeit ist die so genannte Burst-Verarbeitung von Paketen. Hierbei sendet und empfängt eine Anwendung nicht immer nur ein einzelnes Paket. Stattdessen versucht die Anwendung stets eine Menge von Paketen zu versenden oder zu empfangen. Hierdurch wird die Anzahl der Kontext-Wechsel auf Kosten einer eventuell höheren Verzögerung bei der Paketverarbeitung reduziert.

(3) Ein dritter Ansatz ist das Vermeiden von Unterbrechungen bei der Verarbeitung von Paketen. Eine Möglichkeit hierzu sind so genannte Polling-Mode Treiber für Netzwerkschnittstellen. Hierbei werden nicht mehr Unterbrechungen beim Empfang von Paketen ausgelöst. Stattdessen überprüft der Netzwerk-Treiber aktiv, ob neue Pakete von der Netzwerkschnittstelle empfangen worden sind.

(4) Ein vierter Ansatz ist das Vermeiden von Seiteneffekten anderer Anwendungen. Eine Möglichkeit hierzu ist die exklusive Zuweisung von Anwendungen auf Prozessor-Kerne. Hierbei wird die Paket-verarbeitende Anwendung fest einem Prozessor-Kern zugewiesen und ausschließlich auf diesem ausgeführt. Zusätzlich wird verhindert, dass andere Anwendungen auf dem selben Prozessor-Kern ausgeführt werden können.

(5) Ein fünfter Ansatz ist das Beachten der Computer-Architektur beim Entwurf der Paket-verarbeitenden Anwendung. Beispiele hierfür sind ein NUMA-gewahrer Entwurf und das Beachten von *HyperThreading*. Beim NUMA-gewahren Entwurf wird dafür gesorgt, dass die Paket-verarbeitende Anwendung ausschließlich Arbeitsspeicher verwendet, der an den Prozessor angeschlossen ist, auf dem die Anwendung ausgeführt wird. Ein Beispiel für die Beachtung von *HyperThreading* ist, dass man die Paket-verarbeitende Anwendung exklusiv einem Prozessor-Kern zuweist und zusätzlich verhindert, dass andere Anwendungen auf einem logischen Prozessor-Kern ausgeführt werden, der den selben physikalischen Prozessor-Kern verwendet.

5.3.2 Datenaustausch zwischen Anwendungen und Rahmenwerk

In diesem Abschnitt wird der Datenaustausch zwischen Anwendungen und dem Rahmenwerk auf einem Endsystem behandelt. Da Anwendungen und das Rahmenwerk als reguläre Prozesse oberhalb des Betriebssystems ausgeführt werden, werden Mechanismen zur Inter-Prozess-Kommunikation (IPC-Mechanismen) für den Datenaustausch verwendet. Daher werden zunächst verschiedene IPC-Mechanismen und deren Leistungsfähigkeit vorgestellt. Anschließend werden mögliche Leistungsverbesserungen des Datenaustausch zwischen Anwendungen und dem Rahmenwerk vorgestellt.

5.3.2.1 IPC-Mechanismen in Betriebssystemen

Gängige Betriebssysteme wie Linux bieten verschiedene Mechanismen zur Kommunikation zwischen Prozessen (Inter Process Communication, IPC). Da es sich beim Rahmenwerk und den Anwendungen um verschiedene Prozesse auf den Endsystemen handelt, sind diese IPC-Mechanismen für die Anbindung von Anwendungen an das Rahmenwerk interessant. Daher werden an dieser Stelle zunächst die verschiedenen IPC-Mechanismen vorgestellt und anschließend bezüglich ihre Verwendbarkeit für die Anbindungen von Anwendungen an das Rahmenwerk bewertet. Linux bietet die folgenden IPC-Mechanismen: FIFO, Pipe, Local Socket und Shared Memory (vgl. Inter Process Communication Kapitel in [79]).

PIPE und FIFO

Eine PIPE ist ein unidirektionaler, serieller IPC-Mechanismus. Die Reihenfolge beim Auslesen so wie beim Schreiben von Daten bleibt erhalten. Eine PIPE besteht aus einem Schreib-Ende und einem Lese-Ende. Werden Daten in das Schreib-Ende geschrieben, können sie aus dem Lese-Ende ausgelesen werden. Normalerweise werden PIPEs zwischen zwei Threads des selben Prozesses oder zwischen Eltern- und Kind-Prozessen verwendet. Eine PIPE besitzt eine beschränkte Kapazität. Schreibt ein Prozess in eine PIPE, die bereits voll ist, blockiert der Schreibbefehl, bis wieder Platz in der PIPE verfügbar ist. Somit bietet eine PIPE eine automatische Synchronisation der miteinander kommunizierenden Prozesse. Die Lese- und Schreib-Enden der PIPE werden durch separate Lese- und Schreib- Datei-Deskriptoren repräsentiert. Für die Lese- und Schreibbefehle können reguläre Lese- und Schreiboperationen des Betriebssystems verwendet werden.

Eine PIPE erlaubt nur Kommunikation zwischen verwandten Prozessen (Eltern- und Kind-Prozessen). Außerdem wird nur eine Eins-zu-Eins Kommunikation unterstützt. Das heisst, nur zwei Prozesse können über eine PIPE miteinander Daten austauschen. Die Prozesse müssen darüber hinaus gleichzeitig laufen bzw. gestartet werden, um miteinander kommunizieren zu können. Daher sind PIPEs nicht für die Kommunikation zwischen Anwendungen und dem Rahmenwerk geeignet. Ein so genannter FIFO stellt eine PIPE-Alternative für Prozesse, die nicht verwandt sind, dar.

Eine First In First Out (FIFO) Datei entspricht einer PIPE mit einem Namen im Datei-System. Daher wird ein FIFO auch Named Pipe genannt. Beliebige Prozesse können das Lese- oder Schreib-Ende des FIFO öffnen, also auch Prozesse, die nicht miteinander verwandt sind. Die Kommunikation über einen FIFO verläuft folgendermaßen. Zunächst wird ein FIFO als Datei im Dateisystem angelegt. Daraufhin erfolgt der Zugriff auf den FIFO wie auf eine normale Datei. Ein Prozess öffnet den FIFO als lesende Datei. Ein anderer Prozess öffnet den selben FIFO als schreibende Datei. Dann führen die Prozesse reguläre Lese- und Schreiboperationen des Betriebssystems auf dem FIFO aus, um Daten auszutauschen.

Auf einen FIFO können gleichzeitig mehrere Prozesse lesend und mehrere Prozesse schreibend zugreifen. Allerdings werden bei parallelem schreibenden und lesenden Zugriff Mechanismen zur Koordination benötigt. Die Daten müssen in diesem Fall in einer Art strukturiert werden, die es erlaubt, die Daten den unterschiedlichen kommunizierenden Prozessen zuzuordnen (z.B. in Nachrichten mit Quell- und Ziel-Adressen).

Socket und Local Socket

Sockets können auch für die Inter-Prozess-Kommunikation verwendet werden. Sockets bieten bidirektionale Kommunikation zwischen Prozessen und haben folgende Parameter: die Art des Sockets, der Namensraum des Sockets und das Protokoll. Die Art des Sockets

gibt an, ob es sich um einen Connection Socket oder Datagram Socket, also einen verbindungsorientierten oder verbindungslosen Socket, handelt. Der Namensraum des Sockets gibt an, aus welchem Namensraum die verwendeten Adressen stammen, wie z.B. Local Namespace. Das Protokoll gibt das zu verwendende Protokoll, wie z.B. TCP, UDP, UNIX, an.

Der Ablauf bei der Kommunikation über einen Socket ist der Folgende. Zunächst erstellt eine Anwendung einen Socket und gibt dabei die Parameter des Sockets an. Die Erstellung des Sockets liefert einen Datei-Deskriptor, auf den geschrieben bzw. von dem gelesen werden kann mit Hilfe der regulären Lese- und Schreiboperationen des Betriebssystems. Die Kommunikation über Sockets verläuft in der Regel nach dem Client-/Server-Modell. Ein Server wartet auf Client-Verbindungen und die Clients verbinden sich zu dem Server. Bei einem Connection Socket bindet sich der Server an einen Socket. Daraufhin lauscht er auf dem Socket und wartet auf Client-Verbindungen. Bei einer neuen Client Verbindung erstellt der Server einen neuen Socket für die Client-Verbindung. Sockets erlauben somit das Multiplexen mehrerer Clients über separate Sockets für jede Verbindung.

Eine besondere Socket-Variante ist der Local Socket bzw. UNIX-Domain Socket. Diese Variante wird für den Datenaustausch zwischen Prozessen auf dem selben Endsystem verwendet. Local Sockets nutzen den Local Namespace. Zum Verbinden der kommunizierenden Anwendungen werden Dateien im Dateisystem verwendet. Die verwendeten Adressen sind daher Dateinamen.

Shared Memory

Ein weiterer IPC-Mechanismus ist der so genannte Shared Memory bzw. geteilter Speicher. Bei Shared Memory handelt es sich um den vermeintlich schnellsten IPC-Mechanismus. Shared Memory erlaubt mehreren Prozessen, auf den selben Speicher zuzugreifen. Dabei verhält sich der geteilte Speicher wie eigener Speicher des Prozesses. Alle Änderungen der Daten im Speicher sind direkt für alle Prozesse sichtbar. Die Vorteile des gemeinsamen Speichers sind, dass keine Systemaufrufe oder Kontextwechsel benötigt werden. Außerdem wird kein zusätzliches Kopieren der Daten durchgeführt. Ein Nachteil ist allerdings, dass Shared Memory keine Synchronisation der miteinander kommunizierenden Prozesse bietet. Das heisst es sind so genannte Wettlaufsituationen (Race Conditions) möglich, bei denen Prozesse auf die selben Daten schreibend zugreifen und dabei Inkonsistenzen verursachen. Die Synchronisation der Zugriffe auf die Daten muss von Prozessen selbst übernommen werden. Hierfür können die Prozesse Semaphore, Locks etc. verwenden.

Die Kommunikation über Shared Memory läuft folgendermaßen ab. Ein Prozess muss den geteilten Speicher-Bereich allozieren. Andere Prozesse binden sich dann an den geteilten Speicher-Bereich. Wenn Prozesse mit der Kommunikation fertig sind, entfernen sie die Bindung. Am Ende muss ein Prozess den geteilten Speicher-Bereich wieder freigeben.

(1) *Anlegen eines geteilten Speicherbereichs*: Ein Prozess muss den geteilten Speicherbereich anlegen. Dabei wird ein Schlüssel angegeben, der den Speicherbereich identifiziert. Andere Prozesse geben den selben Schlüssel an, um sich an den Speicherbereich anzubinden. Beim Anlegen des Speicherbereichs werden die Größe des Speicherbereichs, Optionen und Zugriffsrechte festgelegt.

(2) *Anbinden an Speicherbereich*: Beim Anbinden an den Speicherbereich werden als Parameter der Identifikator des Speicherbereichs und ein Zeiger auf lokalen Speicher, wohin Speicherbereich angebunden werden soll, angegeben. Zusätzlich werden Optionen angegeben, wie z.B. ob nur lesender Zugriff gewünscht ist.

(3) *Entfernen der Bindung*: Beim Entfernen einer Bindung an einen Speicherbereich wird als Parameter die Adresse des Speicherbereichs angegeben. Handelt es sich bei dem Prozess um den letzten Prozess, der den Speicherbereich nutzt, wird der Speicherbereich wieder freigegeben.

Wie bereits erwähnt ist ein Nachteil bei der Verwendung von Shared Memory, dass ein Protokoll zum Austausch von Daten und Vermeiden von Race Conditions benötigt wird. Außerdem muss der Schlüssel, der einen Speicherbereich identifiziert, ausgetauscht werden.

Leistung

Im Folgenden wird untersucht, welche Leistung bei dem Datenaustausch zwischen Anwendungen mit verschiedenen IPC-Mechanismen erreicht werden kann. Hierfür werden die möglichen Paketraten in Experimenten [98] gemessen. Zwei Anwendungen auf dem selben Endsystem tauschen über verschiedene IPC-Mechanismen Pakete miteinander aus. Außerdem wird die Länge der ausgetauschten Pakete variiert. Das Endsystem verwendet einen Intel i7-2630QM Prozessor mit vier Prozessor-Kernen. Als Betriebssystem wird Slackware Linux 14.1 mit dem Linux Kernel der Version 3.14.16 benutzt. Die beiden Anwendungen tauschen so schnell wie möglich Pakete miteinander aus und messen dabei die Paketrate. Die übertragenen Pakete bestehen aus einem Paketkopf, der die Nutzdatenlänge enthält, und den Nutzdaten selbst.

In Abbildung 5.11 werden die Ergebnisse der Untersuchung dargestellt. Auf der x-Achse ist die Größe der Nutzdaten (*Datengröße*) in Byte abgebildet. Die untersuchten Datengrößen sind 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400 und 1500 Byte. Auf der y-Achse sind die gemessenen Paketraten in Millionen Paketen pro Sekunde (MPPS) zu sehen. Die Kurven zeigen die beim Datenaustausch über verschiedene IPC-Mechanismen erreichten Paketraten. Dabei wird für jede Datengröße der Mittelwert (arithmetisches Mittel) über sechs Versuchsläufe von jeweils 10 Sekunden dargestellt. Die Kurven *dsocket* und *ssocket* zeigen die erreichten Paketraten mit Local Sockets in den Varianten *Datagram Socket* und *Stream Socket*. Die Kurve *pipe* zeigt die Paketrate mit einem FIFO (*Named Pipe*). Die

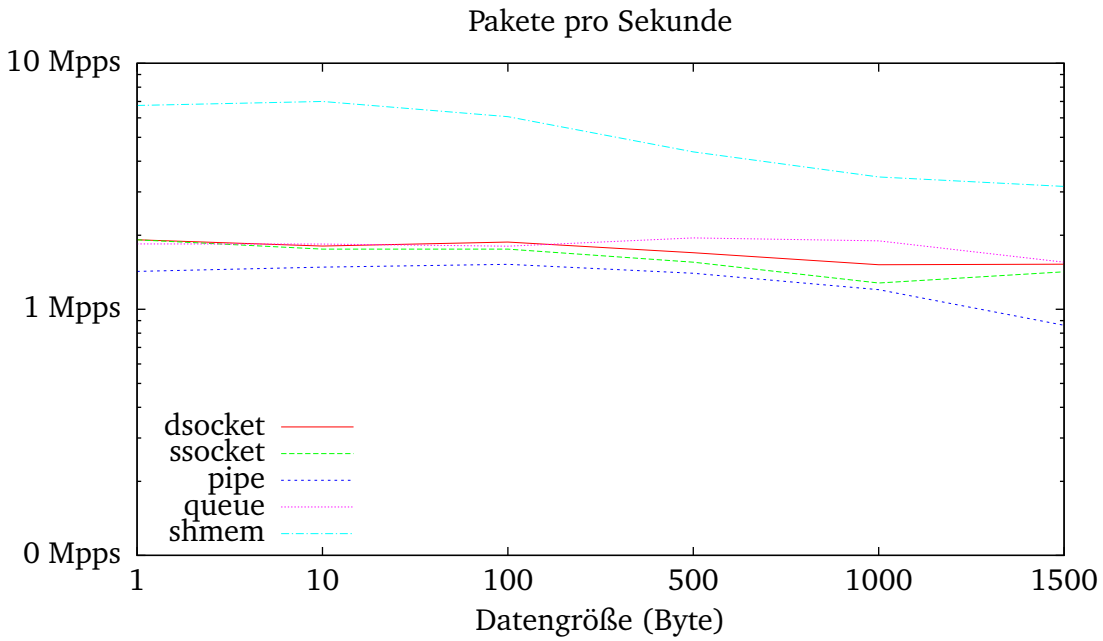

Abbildung 5.11: *Paketraten beim Austausch über IPC-Mechanismen mit variierenden Paketlängen.*

Kurve *queue* zeigt die Ergebnisse mit *POSIX Message Queues*. Die Kurve *shmem* zeigt die Messwerte mit *Shared Memory*. Während die IPC-Mechanismen Local Socket, FIFO und Message Queue nicht mehr als 2 MPPS erreichen, erreicht Shared Memory Paketraten von bis zu 7 MPPS. Shared Memory erzielt damit eine wesentlich höhere Leistung als die anderen IPC-Mechanismen.

5.3.2.2 Optimierungen

Zur Verbesserung der Leistung des Datenaustauschs zwischen Anwendungen und dem Rahmenwerk können ähnliche Optimierungen wie beim Datenaustausch zwischen dem Rahmenwerk und dem Netz verwendet werden. Für hohe Datenraten sollte möglichst auf das Kopieren von Daten und Kontextwechsel zwischen Anwendung und Betriebssystem verzichtet werden. Hierdurch bieten sich Burst-Verarbeitung und Shared Memory an.

Leistung

Im Folgenden wird untersucht, welche Leistungssteigerung beim Nachrichtenaustausch zwischen Anwendungen mit den verschiedenen IPC-Mechanismen durch Optimierungen möglich ist. Hierfür werden die möglichen Paketraten in Experimenten [98] gemessen. Zwei Anwendungen auf dem selben Endsystem tauschen über verschiedene

IPC-Mechanismen so schnell wie möglich Nachrichten miteinander aus und messen dabei die erreichte Paketrate. Außerdem wird die Länge der ausgetauschten Pakete variiert. Für das Experiment wird der selbe Versuchsaufbau wie in den Experimenten in Abschnitt 5.3.2.1 verwendet. Allerdings wird die Optimierung Burst-Verarbeitung verwendet. Anstatt in jeder Sende- oder Empfangsoperation des IPC-Mechanismus, wie in der vorherigen Evaluierung, nur ein Paket zu übertragen, werden in den folgenden Experimenten gleichzeitig immer 32 Nachrichten in einer Operation übertragen.

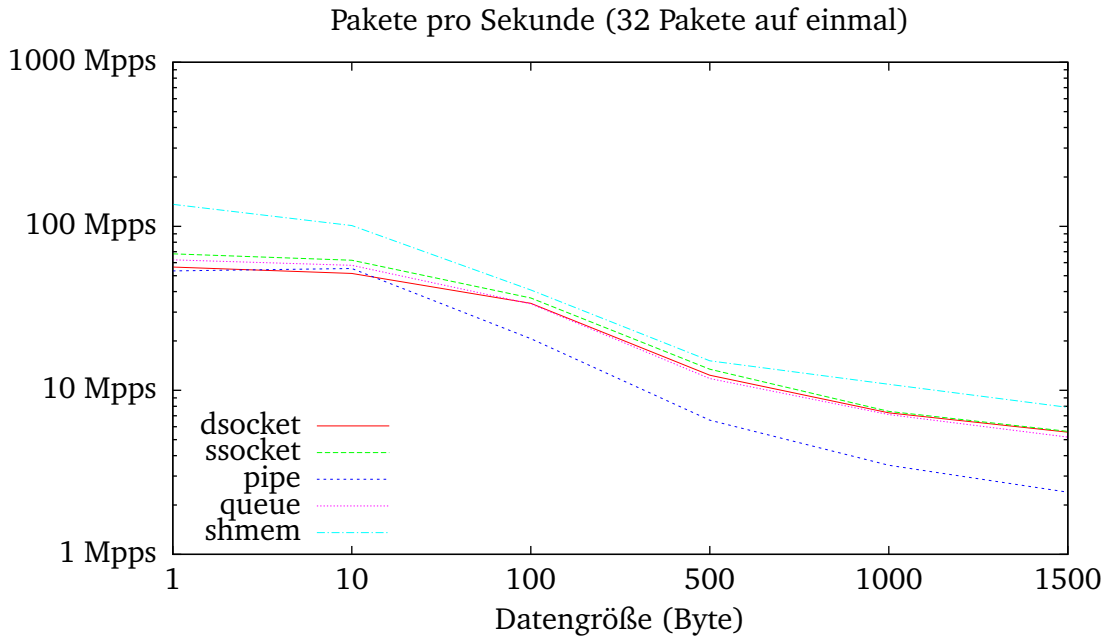

Abbildung 5.12: *Paketraten beim Austausch über IPC-Mechanismen mit Burst-Verarbeitung (32 Nachrichten) mit variierenden Paketlängen.*

In Abbildung 5.12 werden die Ergebnisse der Untersuchung dargestellt. Auf der x-Achse ist die Größe der Nutzdaten (*Datengröße*) in Byte abgebildet. Die untersuchten Datengrößen sind 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400 und 1500 Byte. Auf der y-Achse sind die gemessenen Paketraten in Millionen Pakete pro Sekunde (MPPS) zu sehen. Die Kurven zeigen die beim Datenaustausch über verschiedene IPC-Mechanismen erreichten Paketraten. Dabei wird für jede Datengröße der Mittelwert (arithmetisches Mittel) über sechs Versuchsläufe von jeweils 10 Sekunden dargestellt. Die Kurven *dsocket* und *ssocket* zeigen die erreichten Paketraten mit Local Sockets in den Varianten *Datagram Socket* und *Stream Socket*. Die Kurve *pipe* zeigt die Paketrate mit einem FIFO (*Named Pipe*). Die Kurve *queue* zeigt die Ergebnisse mit *POSIX Message Queues*. Die Kurve *shmem* zeigt die Messwerte mit *Shared Memory*. Durch die Burst-Verarbeitung sind große Leistungssteigerungen möglich. Die IPC-Mechanismen Local Socket und FIFO können um

die 60 MPPS erreichen, was in etwa einer Leistungssteigerung von Faktor 30 gegenüber der Übertragung ohne Burst-Verarbeitung bedeutet. Shared Memory erreicht sogar bis zu maximal 140 MPPS, was einer Steigerung von Faktor 20 gegenüber der Übertragung ohne Burst-Verarbeitung entspricht. Burst-Verarbeitung und Shared Memory stellen somit eine vielversprechende Kombination für einen leistungsfähigen Datenaustausch dar.

Zusammenfassung

Betriebssysteme wie Linux bieten verschiedene Mechanismen zur Kommunikation zwischen Anwendungen. Durch die Wahl eines geeigneten Mechanismus kann eine Leistungssteigerung bei der Inter-Prozess-Kommunikation erreicht werden. Für hohe Datenraten auch bei kleinen Paketlängen sollte auf das Kopieren von Daten und Kontextwechsel zwischen Anwendungen und Betriebssystem so weit wie möglich verzichtet werden. Der für eine hohe Leistung am vielversprechendste IPC-Mechanismus ist daher Shared Memory in Kombination mit Burst-Verarbeitung und sollte auch im Rahmenwerk eingesetzt werden.

5.3.3 Datenverarbeitung innerhalb des Rahmenwerkes

Wenn das Rahmenwerk Nachrichten von Anwendungen oder vom Netz empfängt, müssen diese Nachrichten innerhalb des Rahmenwerkes weiterverarbeitet werden. Um eine möglichst hohe Leistung beim Nachrichtenaustausch zwischen Anwendungen und dem Netz zu erreichen, muss demnach auch das Rahmenwerk eine möglichst hohe Paket- bzw. Datenrate verarbeiten können. In diesem Abschnitt wird die Nachrichtenverarbeitung innerhalb des Rahmenwerkes näher betrachtet.

5.3.3.1 Optimierungen

Es gibt ähnliche Möglichkeiten die Leistung der Nachrichtenverarbeitung im Rahmenwerk zu steigern wie beim Datenaustausch zwischen dem Rahmenwerk und dem Netz. Für möglichst hohe Datenraten sollte soweit möglich auf das Kopieren von Daten und Kontextwechsel zwischen dem Rahmenwerk und dem Betriebssystem verzichtet werden.

Eine weitere Möglichkeit ist die Parallelisierung der Nachrichtenverarbeitung im Rahmenwerk. Hierfür bietet es sich an, die Nachrichtenverarbeitung in mehrere Threads aufzuteilen, so genanntes Multi-Threading. Eine Alternative ist, mehrere Prozesse für die Nachrichtenverarbeitung zu verwenden, so genanntes Multi-Processing. Allerdings haben mehrere Prozesse den Nachteil, dass zur Interaktion zwischen den Prozessen IPC-Mechanismen verwendet werden müssen, die langsamer sein können als ein Datenaustausch zwischen verschiedenen Threads innerhalb des selben Prozesses. Wird die

Ausführung des Rahmenwerkes in mehrere Threads aufgeteilt, sollte auf die Eigenschaften des Endsystems bzw. der zur Verfügung stehenden Prozessoren Rücksicht genommen werden. Zur Vermeidung negativer Cache-Effekte zwischen anderen Anwendungen und dem Rahmenwerk, können Prozessor-Kerne explizit dem Rahmenwerk zugeordnet werden. Diese Prozessor-Kerne dürfen dann von keinen anderen Anwendungen verwendet werden. Verwendet das Endsystem eine Technologie wie *HyperThreading*, muss darauf geachtet werden, dass andere Anwendungen nicht auf logischen Prozessor-Kernen ausgeführt werden, die auf dem selben physikalischen Prozessor-Kern wie das Rahmenwerk ausgeführt werden. Bei mehreren Prozessoren sollte auf so genannte NUMA-Effekte Rücksicht genommen werden. Das Rahmenwerk sollte demnach nur Speicher verwenden, der an den Prozessor angebunden ist, auf dem das Rahmenwerk ausgeführt wird. Außerdem sollten Anwendungen und das Rahmenwerk dementsprechend, soweit möglich, auf dem selben Prozessor ausgeführt werden. Weitere Möglichkeiten sind leistungsoptimierte Datenstrukturen und Funktionen. Es bietet sich beispielsweise an, bei Warteschlangen so weit wie möglich auf teure Synchronisationsmechanismen wie Locks zu verzichten. Außerdem sollten Datenstrukturen verwendet werden, die einen effizienten Zugriff ermöglichen, wie z.B. Hash-Tabellen. Eine weitere Möglichkeit zur Leistungssteigerung ist, bei bedingter Ausführung von Operationen, der Sprungvorhersage des Prozessors Hinweise auf den wahrscheinlichen Ablauf der Operationen zu geben. Müssen Aufgaben zeitlich verzögert ausgeführt oder Zeiten gemessen werden, kann die Leistung dadurch gesteigert werden, das TSC-Register des Prozessors auszulesen, anstatt Timer zu verwenden. Außerdem ist es für eine Steigerung der Leistung sinnvoll, Hardware-optimierte Funktionen, z.B. zum Kopieren von Daten, zu verwenden, anstatt reguläre Funktionen des Betriebssystems.

5.3.4 Zusammenfassung

Durch Optimierungen wie die Burst-Verarbeitung und den direkten Zugriff auf Netzwerkschnittstellen kann eine hohe Leistung beim Datenaustausch zwischen dem Rahmenwerk und dem Netz erreicht werden. Durch geteilten Speicher kann ein leistungsfähiger Datenaustausch zwischen Anwendungen und dem Rahmenwerk erreicht werden. Durch das Vermeiden von Kopien und Parallelisierung kann eine hohe Leistung bei der Datenverarbeitung im Rahmenwerk erreicht werden. DPDK-NENA, das im Rahmen dieser Arbeit entwickelte Rahmenwerk für DsNs, setzt daher diese Optimierungen ein, um eine hohe Leistung zu erreichen.

5.4 Lösungen für schnelle Paketverarbeitung

Aufgrund des großen Optimierungspotenzials bei dem Datenaustausch zwischen Anwendungen und dem Netz sind verschiedene Lösungen zur schnellen Paketverarbeitung entwickelt worden. Diese Lösungen bieten eine Alternative zur Socket-API und erlauben einen direkten Zugriff von Anwendungen auf Netzwerkschnittstellen. Somit ermöglichen sie das direkte Versenden und Empfangen beliebiger Pakete aus Anwendungen mit einer hohen Leistung. Daher ist eine Verwendung einer solchen Lösung in einem Rahmenwerk für DsNs sehr interessant. In diesem Abschnitt werden die Lösungen *Netmap*, *PF_RING* und *DPDK* vorgestellt. Abschließend werden am Ende dieses Abschnitts die Lösungen zusammengefasst und miteinander verglichen. Dabei wird gezeigt, dass die Lösung DPDK den größten Funktionsumfang bei gleicher erwarteter Leistung bietet und daher eine geeignete Lösung für eine Verwendung in einem Rahmenwerk für DsNs darstellt.

5.4.1 Netmap

Netmap [94] ist eine Schnittstelle, die als Ersatz für die Socket-API entwickelt worden ist. Netmap wird für die Betriebssysteme FreeBSD und Linux entwickelt und ist auf Anwendungen ausgelegt, die direkt Ethernet-Rahmen versenden und empfangen können. Netmap verwendet angepasste Netzwerk-Treiber, die die Anzahl der Unterbrechungen reduzieren und die Leistung gegenüber den regulären Treibern steigern sollen. Netmap erreicht eine hohe Leistung hauptsächlich durch die Vermeidung von Systemaufrufen und Unterbrechungen, durch das Vermeiden von Kopien und durch Burst-Verarbeitung.

Die Schnittstelle von Netmap zu den Anwendungen besteht hauptsächlich aus Ring-Puffern in geteiltem Speicher. Netmap bildet die Ring-Puffer der Netzwerkschnittstelle in die Anwendung ab und verwendet dabei geteilten Speicher, um das Kopieren von Daten zu vermeiden. Anwendungen schreiben in die oder lesen aus den durch Netmap bereitgestellten Ring-Puffer der Netzwerkschnittstelle und informieren Netmap gegebenenfalls über neue Pakete.

Das Senden von Paketen mit Netmap verläuft folgendermaßen. Die Anwendung schreibt Pakete in einen Sende-Ring-Puffer. Daraufhin wird in dem Ring-Puffer das Feld für die freien Einträge aktualisiert. Abschließend informiert die Anwendung den Betriebssystem-Kern über die Änderung des Ring-Puffers bzw. über neue Pakete im Ring-Puffer mittels eines Systemaufrufs. Netmap und der angepasste Treiber versenden die Pakete über die Netzwerkschnittstelle und geben die Einträge im Ring-Puffer wieder frei.

Zum Empfangen von Paketen mit Netmap liest eine Anwendung die Pakete aus einem Empfangs-Ring-Puffer der Netzwerkschnittstelle aus. Hierzu fragt die Anwendung zunächst die Anzahl vorhandener Einträge im entsprechenden Ring-Puffer aus der ab. Über einen Systemaufruf wird dieses Feld in dem Ring-Puffer aktualisiert. Außerdem synchronisiert Netmap bei einem Systemaufruf den Ring-Puffer mit der Netzwerkschnittstelle.

Sind Pakete im Ring-Puffer vorhanden, liest sie die Anwendung aus dem Ring-Puffer. Um die Einträge im Ring-Puffer wieder freizugeben, aktualisiert die Anwendung das Feld für freie Einträge im Ring-Puffer. Netmap kann explizit über neue freie Einträge mit einem Systemaufruf oder implizit beim Ausführen anderer Systemaufrufe (z.B. beim Senden von Paketen) informiert werden.

Pakete werden in Speicher abgelegt, der bei der Initialisierung von Netmap angelegt wird. Dadurch wird bei der Paketverarbeitung kein dynamisches Speicher-Allozieren oder De-Allozieren pro Paket benötigt. Eintreffende Pakete werden über Verweise in den Ring-Puffern von Anwendungen ausgelesen. Wenn ein Paket aus einem Ring-Puffer gelesen worden ist, wird der Speicherplatz im Ring-Puffer wieder freigegeben und kann für nachfolgende Pakete verwendet werden. Eine Anwendung, die Pakete für weitere Verarbeitungen vorhalten muss, muss daher selbst dafür sorgen, dass die Pakete gespeichert werden.

Zum Erreichen einer hohen Leistung vermeidet Netmap soweit möglich Systemaufrufe, und damit den Kontextwechsel zwischen Anwendung und Betriebssystem-Kern. Werden Systemaufrufe ausgeführt, wird versucht die Kosten des Systemaufrufs zu amortisieren. Beispielsweise synchronisiert Netmap die Ring-Puffer mit der Netzwerkschnittstelle bei einem Systemaufruf. Außerdem wird Burst-Verarbeitung ermöglicht. Beim Empfangen können Anwendungen alle verfügbaren Pakete im Ring-Puffer auslesen. Beim Senden können Anwendungen alle freien Einträge im Ring-Puffer mit Paketen füllen. Das Kopieren von Daten wird durch den von Anwendungen und Netmap gemeinsam genutzten Speicher vermieden.

Diese Optimierungen erlauben Netmap, 10 Gigabit/s Netzwerkschnittstellen auch mit Paketen minimaler Länge auszulasten [94]. Netmap wurde außerdem für die Implementierung des Software-Switches VALE [95] verwendet und für die Implementierung eines Hochleistungswebservers für statische Web-Inhalte namens Sandstorm [67] genutzt. In diesen Arbeiten konnte bestätigt werden, dass Netmap eine hohe Leistung erreichen kann.

5.4.2 PF_RING

PF_RING [22, 83] ist als Erweiterung der Socket-API entwickelt worden. PF_RING wird für Linux entwickelt und ist für Anwendungen ausgelegt, die direkt Ethernet-Rahmen empfangen und senden können. PF_RING verwendet ein Linux Kernel Modul, das schnelles Kopieren von Paketen in Ring-Puffer ermöglicht. Zusätzliche unterstützt es angepasste Treiber, die das Kopieren von Paketen in die Ring-Puffer ermöglichen, ohne die Datenstrukturen des Linux Kernels zu durchlaufen. PF_RING stellt einen neuen Socket-Typ namens PF_RING bereit. Über den Socket ist die Kommunikation zwischen Anwendungen und dem Kernel Modul von PF_RING möglich. Die Anwendung erhält ein PF_RING Handle, das an eine physikalische Netzwerkschnittstelle, einen Ring-Puffer

bei Netzwerkschnittstellen mit mehreren Ring-Puffern oder ein virtuelles “any”-Gerät gebunden ist.

Beim Auslesen von Paketen durch die Anwendung werden Pakete von einem Ring-Puffer gelesen, der bei der Initialisierung angelegt wird. Dadurch wird kein dynamisches Speicher-Allozieren oder De-Allozieren pro Paket bei der Paketverarbeitung benötigt. Eintreffende Pakete werden in den Ring-Puffer kopiert und von Anwendungen ausgelesen. Wenn ein Paket aus dem Ring-Puffer gelesen worden ist, wird der Speicherplatz im Ring-Puffer wieder freigegeben und kann für nachfolgende Pakete verwendet werden. Eine Anwendung, die Pakete für weitere Verarbeitungen vorhalten muss, muss daher selbst dafür sorgen, dass die Pakete gespeichert werden.

PF_RING erlaubt es, Filter-Regeln anzugeben. Basierend auf diesen wird das Filtern eintreffender Pakete vom Kernel Modul durchgeführt. Mit entsprechender Unterstützung in der Hardware der Netzwerkschnittstelle ist dabei auch das Filtern in Hardware möglich. Die Filter-Regeln sind mit Aktionen verknüpft, die ausgeführt werden, wenn ein Paket mit der Filter-Regel übereinstimmt. Mögliche Aktionen sind, das Paket zur Anwendung weiterzureichen, das Paket nicht zur Anwendung weiterzureichen, oder das Paket zu reflektieren. Das Reflektieren bedeutet, das Paket über eine andere Netzwerkschnittstelle als der eingehenden Netzwerkschnittstelle ohne weitere Interaktion mit der Anwendung zu versenden.

PF_RING erlaubt zusätzlich so genanntes Balancing und Clustering. Dabei verbinden sich verschiedene Anwendungen zum PF_RING Socket und geben die selbe Cluster ID an. Damit gehören die Anwendungen zu einem Verbund von Anwendungen (Cluster). Der eintreffende Paketstrom wird partitioniert und die verschiedenen Teile des Paketstroms werden an die verschiedenen Anwendungen im Cluster weitergereicht. Das Verteilen des Paketstroms (Balancing) auf die Anwendungen des Clusters kann auf zwei Arten erfolgen: Per-Flow und Round-Robin. Per-flow bedeutet, dass alle Pakete, die zum selben Flow gehören (identifiziert über 5-Tupel), an die selbe Anwendung weitergereicht werden. Bei Round-Robin werden die Pakete reihum an Anwendungen des Clusters weitergereicht. Bei der Wahl einer der beiden Möglichkeiten muss zwischen dem Erhalten der Anwendungslogik (Per-Flow) und evtl. gleichmäßigerer Verteilung der Last auf Anwendungen (Round-Robin) abgewägt werden.

Um die Leistung weiter zu verbessern, unterstützt PF_RING verschiedene angepasste Treiber. So genannte PF_RING-gewahre Treiber schreiben Pakete direkt in Ring-Puffer ohne die Paketverarbeitung des Kernels zu durchlaufen. TNAPI Treiber arbeiten mit verschiedenen parallelen Kernel-Threads, wobei sie nur den Empfang von Paketen ermöglichen. Direct NIC Access (DNA) Treiber erlauben den direkten Datenaustausch mit Anwendungen ohne CPU-Overhead. DNA-Treiber umgehen den Betriebssystem-Kern und das PF_RING Module mit einem Zero-Copy-Ansatz. Somit erlauben sie einen direkten Zugriff auf die Ring-Puffer der Netzwerkschnittstelle. Entsprechend ist dadurch allerdings kein Filtern der Pakete im Kernel möglich. Um den Zugriff von Anwendun-

gen auf Netzwerkschnittstellen mit dem DNA-Treiber zu vereinfachen bietet PF_RING außerdem die Bibliothek Libzero. Libzero unterstützt den Zugriff auf DNA-Treiber aus Anwendungen und so genannte DNA Cluster und DNA Bouncer. DNA Cluster entsprechen dem zuvor beschriebenen Packet Clustering und Balancing mit Zero Copy. Eine eigene Implementierung von *Receive Side Scaling* erlaubt, Pakete auf Warteschlangen der Netzwerkschnittstelle zu verteilen. Dabei kann das Filtern, Verteilen und Duplizieren von Paketen auf mehrere Threads und Anwendungen durchgeführt werden. DNA Bouncer ermöglicht ein direktes Weiterleiten von Paketen zwischen Schnittstellen mit Zero Copy. Es erlaubt dabei dem Nutzer, eine Funktion festzulegen, die entscheidet, ob ein Paket weitergeleitet wird oder nicht. Eine Weiterentwicklung der bisher erwähnten PF_RING-Ansätze sind PF_RING-gewahre Treiber mit Zero-Copy Unterstützung. Sie sind eine Mischung von PF_RING-gewahren Treibern mit bedarfsgesteuertem Zero-Copy. Bei Verwendung von Zero-Copy wird die Netzwerkschnittstelle nicht mehr für den Betriebssystem-Kern verfügbar. Wenn die Netzwerkschnittstelle nicht mehr mit Zero-Copy verwendet wird, wird die Netzwerkschnittstelle wieder für den Betriebssystem-Kern verfügbar. Zero-Copy-Treiber erlauben die gleiche Leistung wie DNA-Treiber.

Durch den direkten Zugriff auf die Netzwerkschnittstellen und die Zero-Copy-Ansätze kann PF_RING eine hohe Leistung erreichen. Dank dieser Optimierungen ist es PF_RING möglich, 10 Gigabit/s mit beliebigen Paketgrößen zu erreichen [83].

5.4.3 DPDK

Das Data Plane Development Kit (DPDK), dokumentiert in [49, 51, 50, 52], von Intel bietet eine Schnittstelle und eine Menge von Programm-Bibliotheken zur schnellen Paketverarbeitung in Anwendungen. DPDK wird für die Betriebssysteme Linux und FreeBSD entwickelt und ist auf Anwendungen ausgelegt, die direkt Ethernet-Rahmen versenden und empfangen können. An dieser Stelle wird eine Übersicht über DPDK, seine Komponenten und den Ablauf bei der Verwendung von DPDK in Anwendungen gegeben.

Komponenten

DPDK besteht aus einer Vielzahl von Komponenten. An dieser Stelle werden einige ausgewählte Komponenten vorgestellt, die für die schnelle Paketverarbeitung in einem Rahmenwerk für DsNs relevant sind.

DPDK nutzt so genannte Polling-Mode Treiber. Hierbei handelt es sich um Polling-basierte Treiber für Netzwerkschnittstellen. Im Gegensatz zu einem Interrupt-getriebenen Treiber werden beim Eintreffen neuer Pakete keine Unterbrechungen ausgelöst. Stattdessen wird über den Treiber explizit abgefragt, ob neue Pakete über die Netzwerkschnittstelle eingetroffen sind. Der Polling-Mode Treiber in DPDK erlaubt einen direkten

Zugriff aus einer Anwendung. Somit können aus einer Anwendung direkt Pakete über die entsprechende Netzwerkschnittstelle gesendet und empfangen werden.

Für die Paketverarbeitung wichtige Datenstrukturen sind die so genannten Message-Buffers und Memory-Pools. Message-Buffers sind mit Meta-Daten versehene Puffer, die für Nachrichten genutzt werden. Message-Buffers werden in Memory-Pools organisiert. Dabei handelt es sich um einen Speicher-Bereich, in dem eine Menge von Message-Buffers angelegt wird.

Eine weitere wichtige Datenstruktur sind die so genannten Shared Memory Rings. Shared Memory Rings sind Ring-Puffer, die zum Austausch und Puffern von Nachrichten genutzt werden können. Diese Puffer können zur Inter-Prozess-Kommunikation zwischen verschiedenen DPDK-Anwendungen genutzt werden.

DPDK abstrahiert von den Prozessor-Kernen, die auf dem Endsystem zur Verfügung stehen, durch so genannte Logical Cores. Unter Linux entsprechen diese logischen Prozessor-Kerne Threads, die an bestimmte Prozessor-Kerne gebunden sind. Mit Hilfe der Logical Cores vereinfacht es DPDK, explizit Prozessor-Kerne der Nachrichtenverarbeitung zuzuweisen.

DPDK legt seine Datenstrukturen in Speicherbereichen ab, die zwischen DPDK-Anwendungen geteilt werden können. Somit kann das Kopieren von Daten beim Datenaustausch zwischen DPDK-Anwendungen vermieden und eine hohe Leistung erreicht werden.

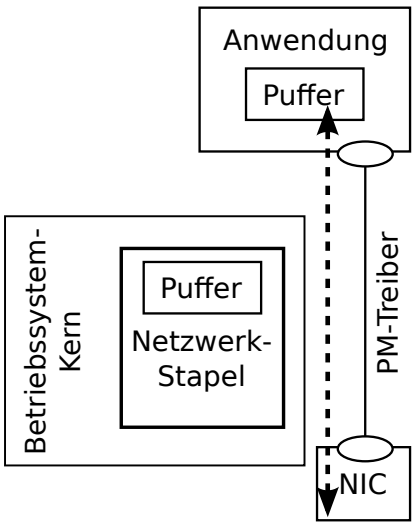

Abbildung 5.13: *Verwendung von DPDK zur Kommunikation.*

Die Folgerung der Verwendung von DPDK in einer Anwendung ist, wie in Abbildung 5.13 dargestellt, dass der Datenaustausch mit der Netzwerkschnittstelle am Betriebssystem vorbei erfolgt. Die Anwendung erhält und versendet direkt Ethernet-Rahmen über die Netzwerkschnittstelle. Die komplette Paketverarbeitung wird demnach in der

Anwendung durchgeführt. Hierdurch können beliebige Protokolle in der Anwendung realisiert werden.

Ablauf

Der Ablauf bei der Ausführung einer DPDK-Anwendung ist der folgende. Beim Start der Anwendung wird der so genannte Environment Abstraction Layer (EAL) ausgeführt. Der EAL ist eine Laufzeitumgebung, die unter anderem den von DPDK verwendeten Speicher reserviert und die Logical Cores initialisiert. Nach dem Start des EAL wird der eigentliche Code der Anwendung ausgeführt. In der Regel erfolgt zunächst die Initialisierung der Anwendung. Dabei werden Memory Pools, Ring-Puffer, die Netzwerkschnittstellen inkl. Treiber und die Datenstrukturen initialisiert. Anschließend erfolgt die Ausführung der Aufgaben der Anwendung auf den Logical Cores, die der Anwendung zur Verfügung stehen. Die Aufgaben umfassen das Empfangen oder Senden von Paketen über Netzwerkschnittstellen, das Empfangen oder Senden von Paketen von anderen Anwendungen über Ring-Puffer, sonstige Paketverarbeitung oder Timer.

Beim Senden von Paketen nimmt eine DPDK-Anwendung für jedes Paket einen Message Buffer aus einem Memory Pool. In diesen Message Buffer wird das entsprechende Paket geschrieben. Zum Versenden an eine andere Anwendung werden die Pakete in den Ring-Puffer eingefügt, der für die Interaktion der beiden Anwendungen verwendet wird. Zum Versenden der Pakete über die Netzwerkschnittstelle werden die Pakete mit Hilfe der Sende-Funktion des Netzwerk-Treibers versandt. Da nur über Referenzen auf Message Buffer zugegriffen wird, müssen beim Versenden der Pakete keine Kopier-Operationen durchgeführt werden.

Beim Empfangen von Paketen von anderen Anwendungen nimmt die DPDK-Anwendung die entsprechenden Message Buffer aus dem Ring-Puffer, der für die Interaktion zwischen den beiden Anwendungen festgelegt worden ist. Beim Empfangen von Paketen aus dem Netz nutzt die DPDK-Anwendung die Empfangsfunktion des Netzwerk-Treibers. Dieser liefert Message Buffers, die die Pakete enthalten. Da auch hier nur Referenzen auf Message Buffer verwendet werden, müssen beim Empfang von Paketen keine Kopier-Operationen durchgeführt werden.

Existierende Arbeiten und Leistung

Um zu zeigen, dass mit DPDK eine Leistungssteigerung möglich ist, wurde eine neue Implementierung des verbreiteten Software Switch Open vSwitch [85] basierend auf DPDK namens DPDK-OVS [53] begonnen. Darüber hinaus wurde DPDK für die Realisierung der Software Switches NetVM [45] und Cuckoo Switch [131] verwendet. In beiden Arbeiten konnte gezeigt werden, dass DPDK in der Lage ist, 10 Gigabit/s zu erreichen.

5.4.4 Zusammenfassung

An dieser Stelle werden die Lösungen für schnelle Paketverarbeitung zusammengefasst und miteinander verglichen. Tabelle 5.1 fasst die Ergebnisse zusammen. In der ersten Spalte werden die Lösungen gelistet. In der zweiten Spalte wird dargestellt, ob es mit der Lösung möglich ist, eine 10 Gigabit/s Schnittstelle bei minimaler Paketlänge von 64 Byte auszulasten. Die dritte und vierte Spalte geben die von der Lösung unterstützen Betriebssysteme und Netzwerkschnittstellen an. Die letzte Spalte gibt den Funktionsumfang der Lösung an. Im Folgenden werden die einzelnen Punkte näher betrachtet.

Tabelle 5.1: *Lösungen für schnelle Paketverarbeitung*

Lösung	10Gbit/s mit minimaler Paketgröße	Unterstützte Betriebssysteme	Unterstützte NIC	Funktionsumfang
Netmap	Ja	Linux, FreeBSD	Intel, Realtek	API
PF_RING	Ja	Linux	Intel, diverse	API + Hilfsbibliothek
DPDK	Ja	Linux, FreeBSD	Intel	API + Bibliotheken

Alle drei Ansätze versprechen eine hohe Leistung. Bei allen Ansätzen kann die Leistung von 10 Gigabit/s Ethernet-Schnittstellen erreicht werden. Daher stellt die Leistungsfähigkeit der Lösungen kein Entscheidungskriterium dar, welche Lösung geeigneter für ein Rahmenwerk für DsNs ist.

PF_RING unterstützt Linux. Netmap und DPDK unterstützen beide Linux und FreeBSD. Damit unterstützen alle drei Ansätze Linux, welches der im Rahmen dieser Arbeit verwendeten Entwicklungsplattform entspricht. Daher stellt die Betriebssystem-Unterstützung kein Entscheidungskriterium dar, welche Lösung am geeignetsten für ein Rahmenwerk für DsNs ist.

Alle Ansätze setzen für hohe Leistung auf angepasste Netzwerk-Treiber. Alle drei Lösungen unterstützen die Intel 1 Gigabit/s und 10 Gigabit/s Netzwerkschnittstellen. Netmap und PF_RING unterstützen noch weitere Netzwerkschnittstellen, aber für maximale Leistung werden die Intel Netzwerkschnittstellen empfohlen. Außerdem sind die Intel Netzwerkschnittstellen sehr verbreitet und standen im Rahmen der Durchführung dieser Arbeit zur Verfügung. Daher stellt die Unterstützung von Netzwerkschnittstellen auch kein Entscheidungskriterium dar, welche Lösung am geeignetsten für ein Rahmenwerk für DsNs ist.

Da die anderen Aspekte kein Entscheidungskriterium für eine der Lösungen liefern, wird der Funktionsumfang der Lösungen betrachtet. Dabei wurde untersucht, was die verschiedenen Ansätze an Funktionalitäten bieten. Bei Netmap und PF_RING müssen Pakete, um sie für eine zukünftige Verarbeitung zu puffern, von der Anwendung kopiert werden, was durch die Kopier-Operation die Leistung reduzieren kann. Bei DPDK bleiben Pakete erhalten und müssen explizit wieder freigegeben werden. Netmap und PF_RING bieten selbst keine Hilfsmittel für weitere Leistungsoptimierungen oder für eine Vereinfachung der Programmierung, wie z.B. explizite Zuweisung von Threads auf CPU-Kerne, Speicher-Verwaltung inkl. Vermeidung von NUMA-Effekten, Warteschlangen und Inter-Prozess-Kommunikation. DPDK bietet viele Programmier-Bibliotheken, die diese Aufgaben vereinfachen. Somit bietet DPDK einen deutlich größeren Funktionsumfang als die anderen Lösungen.

Aufgrund des wesentlich höheren Funktionsumfangs bei erwarteter gleicher Leistung und sowohl ausreichender Betriebssystem- als auch Netzwerkschnittstellen-Unterstützung stellt DPDK eine geeignete Lösung für die Realisierung eines Rahmenwerkes für DsNs mit hoher Leistung dar.

Kapitel 6

Leistungsfähiges Rahmenwerk für Dienst-spezifische Netze

Bei der Verwendung Dienst-spezifischer Netze (DsN) sollten keine Leistungsengpässe auf einem Endsystem entstehen und es sollte eine möglichst hohe Leistung erreicht werden können. Zum Beispiel sollte ein auf hohe Datenraten optimiertes Protokoll auch eine hohe Datenrate erzielen können. Außerdem sollte der Zugriff von Endsystemen auf DsNs keinen Flaschenhals darstellen. Eine besondere Herausforderung ist, dass Nutzer hohe Datenraten unabhängig von der Länge der übertragenen Nachrichten erhalten sollten. Allerdings können gerade kurze Nachrichten dabei ein Problem darstellen. In diesem Kapitel wird daher DPDK-NENA, ein auf Leistung optimiertes Rahmenwerk für DsNs vorgestellt, das hohe Datenraten selbst bei kleinen Paketlängen erreicht. Das Rahmenwerk wird mit Hilfe des Data Plane Development Kits (DPDK) implementiert und setzt verschiedene Optimierungen ein, um eine hohe Leistung zu erreichen. Beispielsweise greift das Rahmenwerk direkt auf die Netzwerkschnittstellen des Endsystems zu und Nutzer können Nachrichten über DsNs versenden oder empfangen, ohne die entsprechenden Daten kopieren zu müssen. Darüber hinaus werden Prozessor-Kerne explizit der Paketverarbeitung zugeordnet, ohne dabei sämtliche Prozessor-Kerne des Endsystems zu belegen. Außerdem wird eine angepasste Hash-Tabelle eingesetzt, die es erlaubt, beliebige einkommende Pakete ohne vorheriges Wissen über die genauen Paketformate und -Inhalte effizient den Verbindungen zu DsNs zuzuordnen. Darüber hinaus erreicht das Rahmenwerk Flexibilität, indem Dienst-spezifische Protokoll-Stapel und Virtuelle Verbindungen zu DsNs in Modulen realisiert werden, die während der Laufzeit hinzugefügt und entfernt werden können. In Experimenten wird die Leistung des Rahmenwerkes evaluiert und mit einer auf der weit verbreiteten Socket-API basierenden Lösung verglichen. Dabei wird gezeigt, dass das Rahmenwerk in der Lage ist, eine mehr als acht mal höhere Leistung als die auf der Socket-API basierende Lösung zu erreichen und eine 10 Gigabit/s

Netzwerkschnittstelle auch mit kleinen Paketlängen auszulasten, was mit der auf der Socket-API basierenden Lösung nicht möglich ist. Das Rahmenwerk kombiniert somit die Flexibilität von DsNs mit einer hohen Leistung und stellt eine geeignete Lösung für die Nutzung von DsNs auf Endsystemen dar.

Gliederung

Dieses Kapitel ist wie folgt gegliedert. In Abschnitt 6.1 wird näher auf die Problemstellung eingegangen und gezeigt, dass für Endsysteme ein flexibles und leistungsfähiges Rahmenwerk benötigt wird. In Abschnitt 6.2 werden relevante Arbeiten vorgestellt. In Abschnitt 6.3 wird DPDK-NENA, ein flexibles und leistungsfähiges Rahmenwerk für DsNs auf Endsystemen, vorgestellt. In den darauf folgenden Abschnitten wird näher auf DPDK-NENA eingegangen. In Abschnitt 6.4 wird die Anwendungskomponente und die Interaktion mit Anwendungen genauer vorgestellt. In Abschnitt 6.5 wird die Basiskomponente und der Datenaustausch über das Netz gezeigt. In Abschnitt 6.6 wird auf die Parallelisierung der Datenverarbeitung eingegangen. In Abschnitt 6.7 wird die Implementierung eines Prototypen von DPDK-NENA vorgestellt. In Abschnitt 6.8 wird DPDK-NENA anhand des Prototypen evaluiert. In Abschnitt 6.9 werden schließlich die Ergebnisse dieses Kapitels zusammengefasst.

6.1 Problemstellung

Für eine flexible und leistungsfähige Nutzung von DsNs wird ein Rahmenwerk für Endsysteme benötigt, das Virtuelle Verbindungen zu DsNs und die entsprechenden Dienst-spezifischen Protokoll-Stapel betreiben und einen schnellen Datenaustausch zwischen den Anwendungen und den DsNs erreichen kann. Diese Problemstellung wird anhand des Beispiels in Abbildung 6.1 verdeutlicht.

In der Abbildung wird ein Endsystem mit einem Nutzer und zwei Anwendungen *Ax* und *Ay* dargestellt. Das Endsystem unterhält für die beiden Anwendungen die zwei Virtuellen Verbindungen *VV1* und *VV2*, über die die beiden Dienst-spezifischen Protokoll-Stapel *DPS1* und *DPS2* verwendet werden. Eine Virtuelle Verbindung zu einem DsN wird in der Abbildung detailliert dargestellt: Die Virtuelle Verbindung *VV1* wird über einen Virtuellen Knoten aufgebaut, der auf einem Infrastruktur-Knoten (*IKz*) ausgeführt wird. Die Konnektivität vom Endsystem zum Infrastruktur-Knoten wird über eine Infrastruktur-Verbindung des Endsystems und mit Hilfe der Infrastruktur-Protokolle in den Infrastruktur-Protokoll-Stapeln *IPS1* und *IPSz* erreicht. Der Virtuelle Knoten unterhält eine Virtuelle Verbindung zum Endsystem (*VV1*), betreibt den Dienst-spezifischen Protokoll-Stapel *DPS1* und führt eine Dienst-spezifische Anwendung *DA* aus.

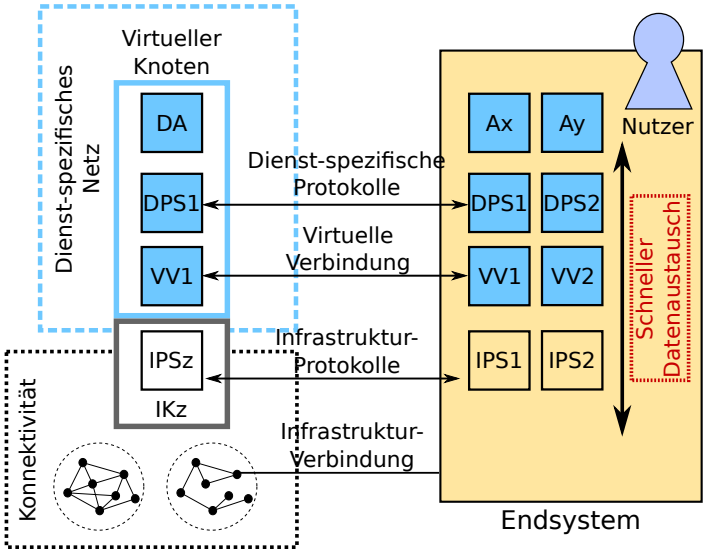

Abbildung 6.1: *Schneller Datenaustausch zwischen Endsystemen und dem Netz.*

Das Endsystem muss für Anwendungen Virtuelle Verbindungen über seine Infrastruktur-Verbindungen mit Hilfe der in den Infrastruktur-Protokoll-Stapeln verfügbaren Infrastruktur-Protokolle aufbauen. Über die Infrastruktur-Verbindungen ausgetauschte Dateneinheiten müssen den Virtuellen Verbindungen zugeordnet werden. Dienst-spezifische Protokoll-Stapel müssen geladen, betrieben und mit den Virtuellen Verbindungen assoziiert werden. Die von den Anwendungen ausgetauschten Nachrichten müssen den Dienst-spezifischen Protokoll-Stapeln zugeordnet werden. Für eine hohe Leistung müssen die Nachrichten von Anwendungen möglichst schnell entgegen genommen, durch die Dienst-spezifischen Protokoll-Stapel verarbeitet, sowie über die Virtuellen Verbindungen und die Infrastruktur-Verbindungen versandt werden. Ebenso müssen über Infrastruktur-Verbindungen empfangene Pakete möglichst schnell den Virtuellen Verbindungen zugeordnet, durch die Dienst-spezifischen Protokoll-Stapel verarbeitet und an die Anwendungen weitergereicht werden.

Auf einem Endsystem wird also ein Rahmenwerk benötigt, in dem die Virtuellen Verbindungen und die Dienst-spezifischen Protokoll-Stapel betrieben werden können. Der Datenaustausch zwischen Anwendungen und dem Netz über das Rahmenwerk muss möglichst schnell verlaufen, um eine möglichst hohe Leistung zu erreichen. Dies beinhaltet einen schnellen Datenaustausch zwischen Anwendungen und dem Rahmenwerk, eine schnelle Datenverarbeitung innerhalb des Rahmenwerkes, sowie einen schnellen Datenaustausch zwischen dem Rahmenwerk und dem Netz. Um die Flexibilität von DsNs zu unterstützen, muss das Rahmenwerk das dynamische Hinzufügen und Entfernen Virtueller Verbindungen und der entsprechenden Protokoll-Stapel ermöglichen.

6.2 Stand der Technik

Wie in Abschnitt 5.1 beschrieben, ist das Erreichen von hohen Datenraten bei kleinen Paketlängen eine Herausforderung. Es existieren einige Beispiele von Protokollen, die kleine Pakete verwenden, wie [103, 130, 63, 23] und Untersuchungen wie in [56, 104, 82, 10, 9] haben gezeigt, dass im Internet ein großer Teil der Pakete klein ist.

Allerdings existiert bisher kein Rahmenwerk für DsNs, das hohe Datenraten beim Datenaustausch zwischen Anwendungen und DsNs erreicht. Beim Entwurf des ursprünglichen NENA, das in Abschnitt 5.2 beschriebene Rahmenwerk für DsNs, stand Leistung nicht im Vordergrund, weswegen NENA nicht für hohe Leistung optimiert ist. In der Evaluation von NENA mit dem ursprünglichen Prototypen [73] wurden Leistungsmessungen durchgeführt. Dabei wurden lediglich geringe Datenraten erreicht. Daher wurde im Rahmen dieser Arbeit DPDK-NENA, ein neues Rahmenwerk für DsNs, entworfen, das hohe Datenraten ermöglicht.

Im Folgenden werden für DPDK-NENA relevante Arbeiten vorgestellt. Wichtig für den Entwurf eines leistungsfähigen Rahmenwerkes für DsNs sind Möglichkeiten, einen schnellen Datenaustausch zwischen Anwendungen und dem Netz zu erreichen. Außerdem existieren Lösungen, die eine leistungsfähige Alternative zur Socket-API bereitstellen. Darüber hinaus sind existierende leistungsfähige Software-Switch Lösungen und Hardware-gestützte Ansätze interessant für diese Arbeit.

6.2.1 Möglichkeiten für schnellen Datenaustausch

Der schnelle Datenaustausch zwischen Anwendungen und DsNs besteht aus (1) einem schnellen Datenaustausch zwischen Anwendungen, (2) einem schnellen Datenaustausch mit dem Netz und (3) einer schnellen Datenverarbeitung. Wie in Abschnitt 5.3 detaillierter beschrieben, existieren verschiedene Möglichkeiten, einen schnellen Datenaustausch zu erreichen.

(1) Der Datenaustausch zwischen Anwendungen auf einem Endsystem verläuft in Betriebssystemen über so genannte IPC-Mechanismen (engl. *Inter Process Communication*). Wie in [79] beschrieben, bieten Betriebssysteme wie Linux verschiedene IPC-Mechanismen an: FIFO, Socket-API, geteilter Speicher. Im Gegensatz zu den anderen IPC-Mechanismen erlaubt geteilter Speicher Anwendungen, direkt Daten miteinander auszutauschen, ohne diese über den Betriebssystem-Kern kopieren zu müssen.

(2) Der Datenaustausch zwischen Anwendungen und Netz verläuft in Betriebssystemen wie Linux, FreeBSD oder Windows in der Regel mit Hilfe der Socket-API. In [94] wurde die Leistung der Socket-API untersucht. Dabei wurde gezeigt, dass die Kontextwechsel zwischen Anwendungen und dem Betriebssystem-Kern und das Kopieren von Daten zwischen Anwendungen und dem Betriebssystem-Kern die Leistung der Socket-API limitieren. Daher wurden in [94] Optimierungen vorgeschlagen und in der Lösung

Netmap integriert. Es wird gezeigt, dass mit Hilfe von Optimierungen des Zugriffs auf Netzwerkschnittstellen wie Polling, mit Bündel-weiser Verarbeitung (Burst-Verarbeitung) von mehreren Paketen zur selben Zeit, durch das Vermeiden von Kontextwechseln und durch das Vermeiden von Kopieren zwischen Anwendungen und Betriebssystem-Kern durch geteilten Speicher Datenraten von heutigen 10 Gigabit/s Netzwerkschnittstellen selbst bei minimalen Paketgrößen erreicht werden können.

(3) Eine Möglichkeit, die Datenverarbeitung auf einem Endsystem zu beschleunigen, ist die Parallelisierung [117]: Durch eine Aufteilung der Paketverarbeitung auf mehrere parallele Threads kann die Leistung gesteigert werden. In [45] werden neben der Parallelisierung weitere Optimierungen verwendet, um die Paketverarbeitung zu beschleunigen. Parallele Threads werden exklusiv an verschiedene Prozessor-Kerne gebunden, so dass die Prozessor-Kerne von keinen anderen Anwendungen bzw. Threads auf dem Endsystem verwendet werden. Außerdem werden Synchronisationsmechanismen wie Locks beim Datenaustausch zwischen Threads und das Kopieren bei der Paketverarbeitung vermieden.

Das in diesem Kapitel beschriebene leistungsfähige Rahmenwerk für DsNs setzt die hier beschriebenen Optimierungen ein, um einen schnellen Datenaustausch zwischen Anwendungen und DsNs zu realisieren.

6.2.2 Lösungen für schnellen Datenaustausch mit dem Netz

Es existieren Lösungen für schnelle Paketverarbeitung, die die Möglichkeiten für einen schnellen Datenaustausch mit dem Netz implementieren und damit eine leistungsfähige Alternative zur Socket-API darstellen. Diese Lösungen können daher als eine Grundlage bei der Implementierung eines leistungsfähigen Rahmenwerkes für DsNs verwendet werden.

Wie in Abschnitt 5.4 detaillierter beschrieben, bieten die Rahmenwerke Netmap [94], PF_RING [22] und DPDK [49] Schnittstellen für einen schnellen Datenaustausch zwischen Anwendungen und dem Netz, bei dem Datenraten von 10 Gigabit/s auch bei kleinen Paketen erreicht werden können. Die Lösungen erlauben einen direkten Zugriff auf die Netzwerkschnittstellen des Endsystems aus Anwendungen heraus. Dadurch können Anwendungen beliebige Pakete über die Netzwerkschnittstellen mit einer hohen Paketrate versenden und empfangen. Dies macht diese Lösungen für DsNs mit ihren Dienst-spezifischen Protokollen besonders interessant. Die Lösungen unterscheiden sich durch ihren Funktionsumfang. Im Gegensatz zu den anderen beiden Ansätzen bietet DPDK eine Vielzahl von Bibliotheken, die die Entwicklung von Anwendungen für schnelle Paketverarbeitung vereinfachen. Aufgrund dessen wird DPDK für die Implementierung des in diesem Kapitel beschriebenen leistungsfähigen Rahmenwerks für DsNs verwendet.

6.2.3 Software-Switches

Es existieren verschiedene Lösungen für Software-basierte Switches, die auf den zuvor beschriebenen Lösungen für schnelle Paketverarbeitung aufbauen wie NetVM [45], CuckooSwitch [131] und VALE [95]. Diese werden in der Regel dazu eingesetzt, Virtuelle Maschinen auf einem Knoten effizient an das Netz anzubinden. Hierfür müssen sie in der Lage sein, Pakete schnell mit Virtuellen Maschinen auszutauschen, diese zu Verarbeitung und schnell mit dem Netz auszutauschen. Dadurch erfüllen sie ähnliche Aufgaben wie ein Rahmenwerk für DsNs auf einem Endsystem. Allerdings muss ein Rahmenwerk für DsNs in der Lage sein, verschiedene Dienst-spezifische Protokolle zu verwenden. Diese Dienst-spezifischen Protokolle führen wiederum beliebige Operationen auf den Paketen aus und können zur Laufzeit hinzugefügt und entfernt werden. Die hier vorgestellten Software-Switches erlauben nicht, die Pakete innerhalb des Software-Switches beliebig zu manipulieren oder gar neue Funktionen zur Paketverarbeitung zur Laufzeit einzuführen. Daher wurden sie nicht für die Realisierung eines leistungsfähigen Rahmenwerkes für DsNs verwendet.

6.2.4 Hardware-gestützte Ansätze

Eine weitere Möglichkeit, die Leistung bei der Paketverarbeitung zu steigern, ist der Einsatz von spezieller Hardware. Beispiele hierfür sind FPGA-basierte Ansätze oder die Verwendung von Graphikkarten.

FPGA-basierte Ansätze wie NetFPGA [65] und Field-Programmable Port Extender (FPX) [64] setzen *Field-Programmable Gate Arrays* (FPGA) ein, um Pakete schnell in Hardware zu verarbeiten. Interessant sind FPGAs, da sie zur Laufzeit neu programmiert werden können. In [64] wird beispielsweise ein programmierbarer Netz-Knoten vorstellt, auf dem zur Laufzeit paketverarbeitende Funktionen wie z.B. Firewalls ausgebracht werden können.

Andere Ansätze machen sich die Eigenschaften moderner Graphikkarten, die aus einer Vielzahl von Verarbeitungseinheiten bestehen, zu Nutze. In PacketShader [38] wird eine Lösung vorgestellt, die Paketverarbeitung auf den Verarbeitungseinheiten von modernen Graphikkarten durchführt und dabei hohe Paketraten erzielt.

Ein Nachteil dieser Ansätze ist, dass sie spezielle Hardware voraussetzen. Außerdem kann die Nutzung der speziellen Hardware auch einen Aufwand mit sich bringen, der den Leistungsgewinn wieder zu Nichte macht. Zum Beispiel benötigt bei der Paketverarbeitung in PacketShader der Wechsel vom Prozessor auf die Graphikkarte Zeit. Dieser Zeitaufwand muss bei der Paketverarbeitung auf der Graphikkarte amortisiert werden. Der auf der Graphikkarte durchgeführte Teil der Paketverarbeitung muss für die Verarbeitung auf der Graphikkarte geeignet sein. Das im Rahmen dieses Kapitels vorgestellte Rahmenwerk DPDK-NENA benötigt keine spezielle Hardware wie FPGAs oder spezielle

Graphikkarten. Allerdings könnten Ansätze mit spezieller Hardware in DPDK-NENA integriert werden, um z.B. bestimmte Funktionen Dienst-spezifischer Protokolle oder des Rahmenwerkes selbst weiter zu parallelisieren oder zu optimieren. Dies wurde allerdings im Rahmen dieser Arbeit nicht weiter verfolgt.

6.2.5 Zusammenfassung

Da bisher kein Rahmenwerk für DsNs existiert, das eine hohe Datenrate beim Datenaustausch zwischen Anwendungen und DsNs erlaubt, wurde im Rahmen dieser Arbeit DPDK-NENA entwickelt. Es setzt auf existierende Optimierungen wie geteilten Speicher, das Vermeiden von Kontextwechseln und Parallelisierung. Darüber hinaus verwendet es DPDK, um direkt auf die Netzwerkschnittstellen des Endsystems zuzugreifen. Im Gegensatz zu existierenden leistungsfähigen Software-Switch Lösungen erlaubt das Rahmenwerk DPDK-NENA, beliebige Dienst-spezifische Protokolle zu betreiben und zur Laufzeit hinzuzufügen oder zu entfernen. Prinzipiell könnte das Rahmenwerk um zusätzliche Hardware-gestützte Ansätze erweitert werden.

6.3 DPDK-NENA

In diesem Abschnitt wird DPDK-NENA, ein leistungsfähiges Rahmenwerk für DsNs vorgestellt. Es wird ein Überblick über die Komponenten von DPDK-NENA, den Betrieb Virtueller Verbindungen sowie Dienst-spezifischer Protokoll-Stapel und den schnellen Datenaustausch über DPDK-NENA gegeben. DPDK-NENA kombiniert die Flexibilität von DsNs mit hoher Leistung.

DPDK-NENA erreicht eine hohe Leistung durch einen direkten Zugriff auf die Netzwerkschnittstellen des Endsystems und einen Datenaustausch zwischen Anwendungen und dem Netz ohne das Kopieren der Daten. Außerdem parallelisiert DPDK-NENA die Datenverarbeitung, indem die Ausführung der verschiedenen Komponenten von DPDK-NENA auf Prozessor-Kerne aufgeteilt wird. DPDK-NENA vermeidet dabei NUMA-Effekte, indem Datenstrukturen im Speicher des selben Prozessors, auf dem DPDK-NENA ausgeführt wird, abgelegt werden.

DPDK-NENA erreicht Flexibilität durch die Verwendung austauschbarer Komponenten. Die austauschbaren Komponenten sind Module, die die Protokolle von DsNs oder Verbindungsmethoden zu DsNs enthalten. Module können zur Laufzeit hinzugefügt und entfernt werden. Außerdem werden die Komponenten von DPDK-NENA generisch gehalten, um zu verhindern, dass sie beim Hinzufügen und Entfernen von Virtuellen Verbindungen oder Dienst-spezifischen Protokollen geändert werden müssen.

Eine Übersicht über DPDK-NENA wird in Abbildung 6.2 dargestellt. Anwendungen wie z.B. *Application 1* und *Application 2* bauen Verbindungen zu Diensten über DPDK-NENA

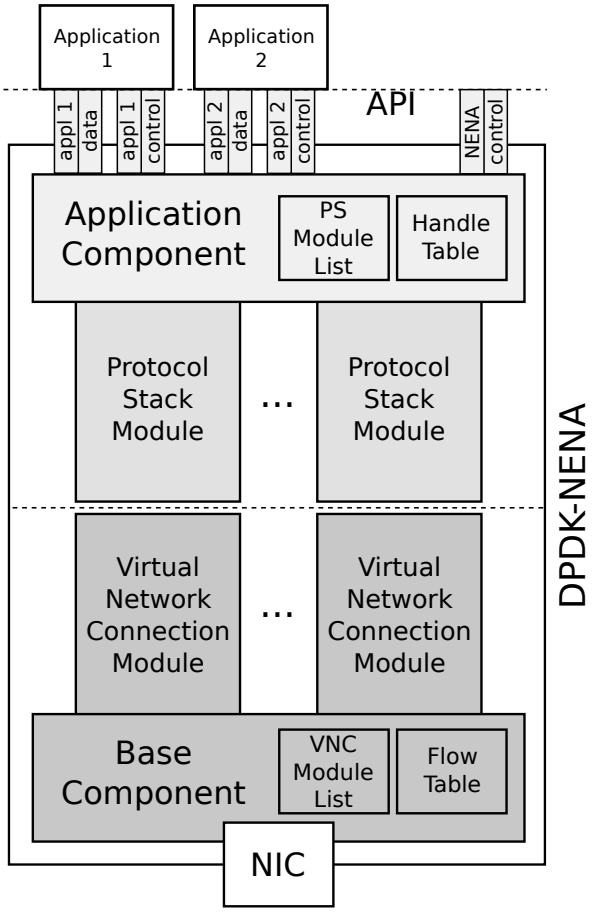

Abbildung 6.2: *Überblick über DPDK-NENA.*

mit Hilfe der *API* auf. DPDK-NENA selbst besteht aus den folgenden Komponenten: eine Anwendungskomponente, Protokoll-Stapel Module, Module für Virtuelle Verbindungen und eine Basiskomponente. Die Anwendungskomponente (*Application Component*) regelt den Zugriff von Anwendungen auf DPDK-NENA und verwaltet die auf dem Endsystem aktiven Protokoll-Stapel Module. Die wichtigsten Datenstrukturen der Anwendungskomponente sind die Liste der aktiven Protokoll-Stapel Module (*PS Module List*) und die *Handle Table*, die Anwendungen den Protokoll-Stapel Modulen zuordnet. Protokoll-Stapel Module (*Protocol Stack Modules*) beinhalten die Dienst-spezifischen Protokolle, die über die Virtuellen Verbindungen zu DsNs betrieben werden. Diese Module können zur Laufzeit hinzugefügt und entfernt werden. Die Module für Virtuellen Verbindungen (*Virtual Network Connection Modules*) beinhalten die Protokolle und Mechanismen, mit deren Hilfe die Virtuellen Verbindungen zu DsNs aufgebaut werden. Auch diese Module können während der Laufzeit hinzugefügt und entfernt werden. Die Basiskomponente (*Base Component*) verwaltet die auf dem Endsystem aktiven Module für Virtuelle Verbindungen und regelt den Zugriff auf die Netzwerkschnittstellen (*NIC*)

des Endsystems. Die wichtigsten Datenstrukturen der Basiskomponente sind die Liste der aktiven Module für Virtuelle Verbindungen (*VNC Module List*) und die *Flow Table*, die vom Netz einkommende Pakete den Modulen für Virtuelle Verbindungen zuordnet.

6.3.1 Flexibler Betrieb Virtueller Verbindungen und Dienst-spezifischer Protokoll-Stapel

DPDK-NENA erreicht Flexibilität durch das Realisieren der Protokoll-Stapel und der Virtuellen Verbindungen in Modulen, die während der Laufzeit hinzugefügt und entfernt werden können. Ein Protokoll-Stapel Modul kann mehreren Modulen für Virtuelle Verbindungen zugeordnet werden. Ein Modul für Virtuelle Verbindungen kann mehreren Protokoll-Stapel Modulen zugeordnet werden. DPDK-NENA verwendet einfache Schnittstellen-Funktionen zu Modulen für Virtuelle Verbindungen und zu Protokoll-Stapel Modulen. Über diese Funktionen werden Nachrichten an die Module übergeben. Während die Module die Nachrichten weiterverarbeiten, können die Module beliebige Operationen auf den Nachrichten ausführen. Die Module für Virtuelle Verbindungen und Protokoll-Stapel Module können in der Implementierung der Schnittstellen-Funktionen selbst entscheiden, ob sie für die Verarbeitung von Nachrichten verwendet werden. Außerdem entscheiden Protokoll-Stapel Module selbst, über welche der ihnen zugeordneten Module für Virtuelle Verbindungen Nachrichten weitergeleitet werden. Genauso entscheiden Module für Virtuelle Verbindungen selbst, an welche der ihnen zugeordneten Protokoll-Stapel Module Nachrichten weitergereicht werden. Dies erlaubt einen flexiblen Nachrichtenfluss durch Module und eine flexible Nachrichtenverarbeitung innerhalb der Module. Im Folgenden wird näher auf die Module für Virtuelle Verbindungen und auf die Protokoll-Stapel Module eingegangen.

6.3.1.1 Virtuelle Verbindungen

Module für Virtuelle Verbindungen enthalten die Protokolle und Methoden, um Virtuelle Verbindungen zu DsNs aufzubauen. Sie können zur Laufzeit geladen und entfernt werden. Jedes Modul stellt die Funktion *canHandle(packet)* bereit. Diese Funktion wird dazu verwendet, um zu überprüfen, ob das Modul für das über den Parameter *packet* angegebene Paket zuständig ist.

Abbildung 6.3 zeigt den Aufbau eines Moduls (*VNC Module*) und dessen Einordnung in DPDK-NENA. Mehrere Protokoll-Stapel Module (*Protocol Stack Modules*) können einem Modul für Virtuelle Verbindungen zugeordnet sein. Das Modul für Virtuelle Verbindungen ist der Basiskomponente (*Base Component*) zugeordnet. Das Modul für Virtuelle Verbindungen besteht aus drei Teilen: die Interaktion mit Modulen für Protokoll-Stapel, die Protokolle und die Interaktion mit der Basiskomponente. Die Interaktion mit Modulen

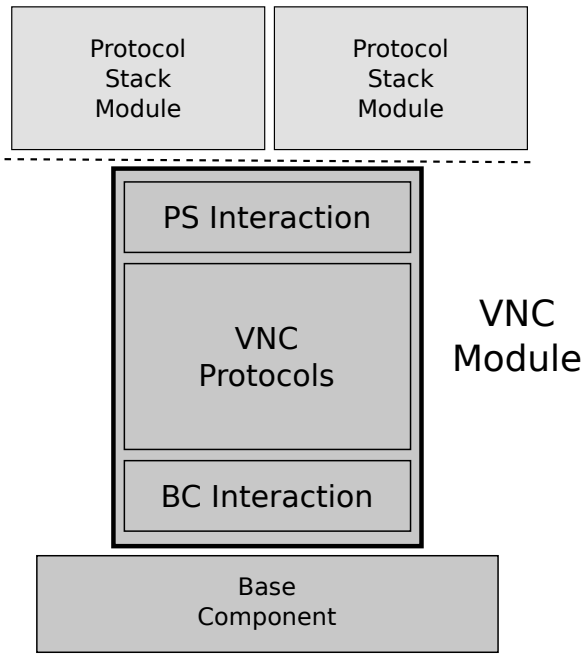

Abbildung 6.3: *Aufbau eines Moduls für Virtuelle Verbindungen und Einordnung in DPDK-NENA.*

für Protokoll-Stapel (*PS Interaction*) enthält die Funktionen, die für den Daten- und Informationsaustausch mit Protokoll-Stapel-Modulen benötigt werden. Die *VNC Protocols* sind die Implementierung der Protokolle, die verwendet werden, um eine Verbindung zu einem DsN aufzubauen. Ein solches Protokoll könnte beispielsweise einen UDP-Tunnel zu einem DsN aufbauen. Die Interaktion mit der Basiskomponente (*BC Interaction*) enthält die Funktionen, die zum Datenaustausch mit dem DsN über die Netzwerkschnittstellen des Endsystems benötigt werden.

Die Schnittstelle eines Moduls für Virtuelle Verbindungen zu DPDK-NENA besteht aus folgenden Funktionen:

- *init*
- *canHandle*
- *processFromNetwork*
- *processFromProtocolStack*
- *processEvent*
- *processProtocolStackChange*
- *processBaseChange*

- *deinitialize*

Die Funktion *init* wird beim Laden des Moduls aufgerufen und ermöglicht dem Modul, interne Datenstrukturen zu initialisieren. *canHandle* wird verwendet, um zu überprüfen, ob ein Modul für ein vom Netz empfangenes Paket zuständig ist. *processFromNetwork* wird aufgerufen, wenn eine Nachricht über das Netz empfangen wird. *processFromProtocolStack* wird aufgerufen, wenn das Modul eine Nachricht von einem Protokoll-Stapel erhält. *processEvent* wird verwendet, um das Modul über ein eingetretenes Ereignis, wie z.B. einen abgelaufenen Timer zu informieren. *processProtocolStackChange* wird verwendet, um dem Modul einen neuen Protokoll-Stapel mitzuteilen bzw. zuzuordnen. *processBaseChange* wird aufgerufen, wenn sich die Anbindung an das Netz ändert. *deinitialize* ist das Gegenstück zur Initialisierungsfunktion und wird aufgerufen, bevor das Modul entfernt wird.

Der Ablauf bei der Verwendung eines Moduls ist der Folgende: Nach dem Laden eines Moduls führt DPDK-NENA einen Aufruf der *init* Funktion durch, um das Modul zu initialisieren. Anschließend kann das Modul verwendet werden. Über den Aufruf der Funktionen *processFromNetwork* und *processFromProtocolStack* verläuft die reguläre Paketverarbeitung durch das Modul. Wird ein Modul nicht mehr benötigt, kann die *deinitialize* Funktion aufgerufen und das Modul entfernt werden.

Ein Modul muss ermitteln, zu welchem Protokoll-Stapel ein vom Netz kommendes Paket gehört. Hierfür muss ein Modul Mechanismen verwenden, die eintreffende Pakete auf Protokoll-Stapel-Module abbilden. Mögliche Mechanismen sind Datenstrukturen für die Abbildung im Modul oder Felder in den übertragenen Paketen. Außerdem muss ein Modul die eintreffenden Pakete in für die Protokoll-Stapel lesbare Nachrichten umwandeln, bevor sie an die Protokoll-Stapel-Module weitergereicht werden.

6.3.1.2 Protokoll-Stapel

Module für Protokoll-Stapel enthalten die Dienst-spezifischen Protokolle, die innerhalb eines DsN benutzt werden. Sie können zur Laufzeit geladen und entfernt werden. Jeder Protokoll-Stapel stellt die Funktion *canHandle(name)* bereit. *canHandle()* wird dazu verwendet, um zu überprüfen, ob ein Protokoll-Stapel für den über den Parameter *name* übergeben Namen zuständig ist.

Abbildung 6.4 zeigt den Aufbau eines Moduls für Protokoll-Stapel (*Protocol Stack Module*) und dessen Einordnung in DPDK-NENA. Das Protokoll-Stapel Modul ist der Anwendungskomponente (*Application Component*) zugeordnet. Mehrere Module für Virtuelle Verbindungen (*Virtual Network Connection Modules*) können dem Protokoll-Stapel-Modul zugeordnet sein. Das Protokoll-Stapel Modul besteht aus drei Teilen: die Interaktion mit der Anwendungskomponente, die Protokolle und die Interaktion mit Modulen für Virtuelle Verbindungen. Die Interaktion mit der Anwendungskomponente (*AC*

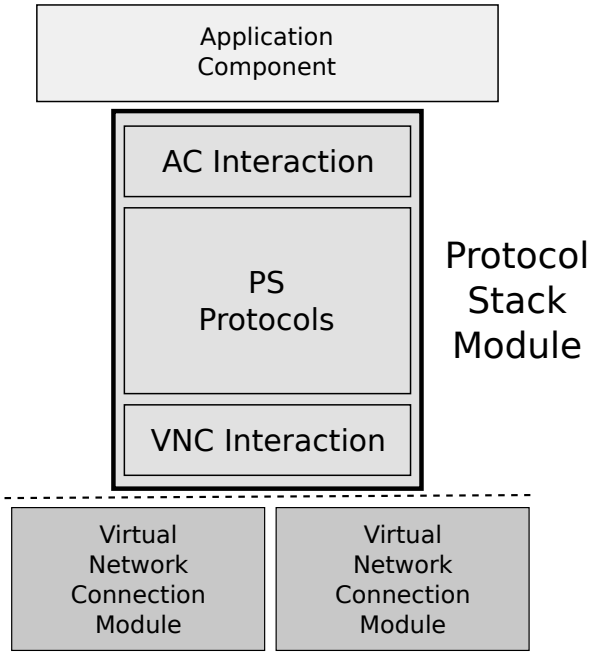

Abbildung 6.4: *Aufbau eines Moduls für Protokoll-Stapel und Einordnung in DPDK-NENA.*

Interaction) enthält die Funktionen, die für den Daten- und Informationsaustausch mit Anwendungen und der Anwendungskomponente benötigt werden. Die *PS Protocols* sind die Implementierung der Dienst-spezifischen Protokolle, die im DsN verwendet werden. Die Interaktion mit Modulen für Virtuelle Verbindungen (*VNC Interaction*) enthält die Funktionen, die zum Datenaustausch mit dem DsN über Virtuelle Verbindungen benötigt werden.

Die Schnittstelle eines Protokoll-Stapel Moduls zu DPDK-NENA besteht aus folgenden Funktionen:

- *init*
- *canHandle*
- *processFromNetwork*
- *processFromApplication*
- *processEvent*
- *processApplicationRequest*
- *processVncChange*
- *deinitialize*

Die Funktion *init* wird beim Laden des Moduls aufgerufen und ermöglicht dem Modul, interne Datenstrukturen zu initialisieren. *canHandle* wird verwendet, um zu überprüfen, ob ein Modul die Anforderungen einer Anwendung erfüllen kann. *processFromNetwork* wird aufgerufen, wenn eine Nachricht über eine Virtuelle Verbindung vom DsN empfangen wird. *processFromApplication* wird aufgerufen, wenn das Modul eine Nachricht von einer Anwendung erhält. *processEvent* wird verwendet, um das Modul über ein eingetretenes Ereignis, wie z.B. einen abgelaufenen Timer zu informieren. *processApplicationRequest* wird verwendet, um dem Modul eine neue Kommunikationsbeziehung einer Anwendung mitzuteilen. *processVncChange* wird aufgerufen, wenn sich die Virtuelle Verbindung zum DsN ändert. *deinitialize* ist das Gegenstück zur Initialisierungsfunktion und wird aufgerufen, bevor das Modul entfernt wird.

Der Ablauf bei der Verwendung eines Moduls ist der Folgende: Nach dem Laden eines Moduls führt DPDK-NENA einen Aufruf der *init* Funktion durch, um das Modul zu initialisieren. Anschließend kann das Modul verwendet werden. Über den Aufruf der Funktionen *processFromApplication* und *processFromNetwork* verläuft die reguläre Paketverarbeitung durch das Modul. Wird ein Modul nicht mehr benötigt, kann die *deinitialize* Funktion aufgerufen und das Modul entfernt werden.

Ein Modul muss ermitteln, zu welcher Kommunikationsbeziehung einer Anwendung eine vom Netz kommende Nachricht gehört. Hierfür muss ein Modul Mechanismen verwenden, die eintreffende Nachrichten auf Kommunikationsbeziehungen abbilden. Mögliche Mechanismen sind Datenstrukturen für die Abbildung im Modul oder Felder in den übertragenen Nachrichten. Außerdem muss das Modul die eintreffende Nachricht in eine für die Anwendung lesbare Nachricht (siehe DPDK-NENA API) umwandeln, bevor die Nachricht an die Anwendung weitergereicht wird.

6.3.2 Schneller Datenaustausch zwischen Anwendungen und Netz

DPDK-NENA erreicht eine hohe Leistung durch einen Datenaustausch zwischen Anwendungen und dem Netz, bei dem Daten nicht kopiert werden müssen, und durch die Parallelisierung der Datenverarbeitung.

In Abbildung 6.5 wird der schnelle Datenaustausch zwischen Anwendungen und dem Netz über DPDK-NENA schematisch dargestellt. Eine Anwendung (*Application*) kommuniziert über DPDK-NENA mit Hilfe von Ring-Puffern in einem geteiltem Speicherbereich (*appl1 data*). Für die Kommunikation wird ein Protokoll-Stapel Modul (*Protocol Stack Module*) und ein Modul für Virtuelle Verbindungen (*Virtual Network Connection Module*) verwendet. Die Anwendungskomponente (*Application Component*) und die Basiskomponente (*Base Component*) nutzen jeweils einen Prozessor-Kern (*CPU-Core*). Das Endsystem verwendet eine Netzwerkschnittstelle (*NIC*). Der Datenaustausch verläuft mit Hilfe von Nachrichten-Puffern, die in einem geteiltem Speicherbereich abgelegt werden (*Message Buffers*). Die durchgezogenen Pfeile stellen den Zugriff auf die Nachrichten-

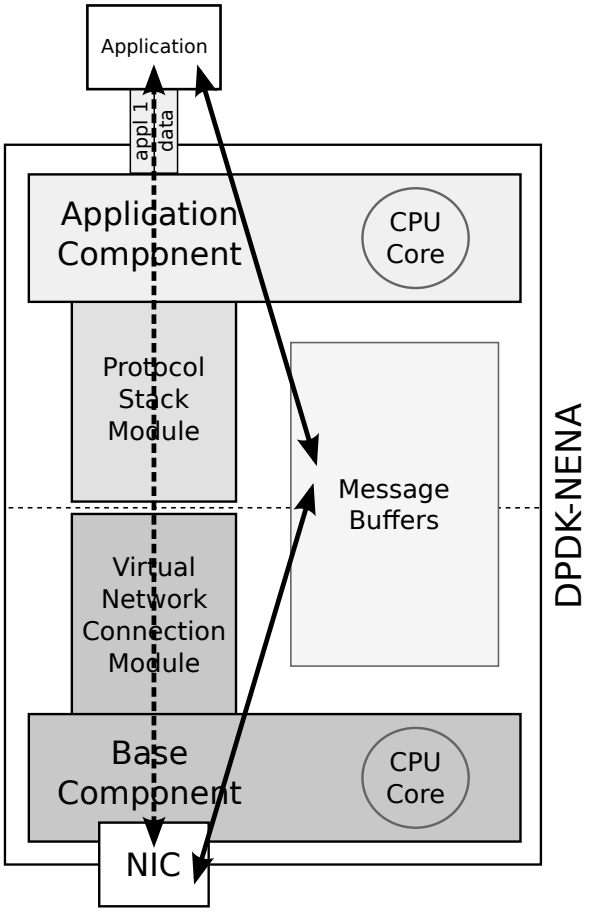

Abbildung 6.5: *Schneller Datenaustausch zwischen Anwendungen und Netz.*

Puffer im geteilten Speicherbereich dar. Der gestrichelte Pfeil stellt die Verarbeitung der Nachrichten in den verschiedenen Komponenten von DPDK-NENA dar.

6.3.2.1 Datenaustausch ohne Kopieren

DPDK-NENA erreicht den Datenaustausch ohne Kopieren folgendermaßen. Die Interaktion mit Anwendungen erfolgt über Ring-Puffer (wie z.B. *appl1 data* in Abbildung 6.5) in zwischen DPDK-NENA und den Anwendungen geteiltem Speicher. Für den Datenaustausch über das Netz greift DPDK-NENA direkt auf die Netzwerkschnittstellen des Endsystems zu. Nachrichten, die zwischen Anwendungen und dem Netz ausgetauscht werden, werden in Nachrichten-Puffern (*Message Buffers*) in einem zwischen DPDK-NENA und Anwendungen geteiltem Speicherbereich abgelegt. Anwendungen greifen direkt auf diesen Speicherbereich zu und können somit direkt Nachrichten in diesen Speicherbereich schreiben oder aus dem Speicherbereich auslesen. DPDK-NENA versendet Nachrichten direkt aus diesem Speicherbereich über die Netzwerkschnittstellen

des Endsystems. Außerdem empfängt DPDK-NENA direkt Nachrichten vom Netz in die Nachrichten-Puffer in diesem Speicherbereich. Die Anwendung und DPDK-NENA tauschen lediglich Referenzen auf Nachrichten in dem geteilten Speicherbereich über die Ring-Puffer miteinander aus. Ebenso werden innerhalb von DPDK-NENA nur Referenzen auf Nachrichten in dem geteilten Speicherbereich ausgetauscht. Wenn Nachrichten manipuliert werden, erfolgt dies direkt auf den Daten im geteilten Speicherbereich.

Für eine kompaktere Schreibweise wird, statt einem *Austausch von Referenzen auf Nachrichten*, im Folgenden meistens die Formulierung *Austausch von Nachrichten* verwendet. Sofern nicht explizit erwähnt, werden in folgenden Beschreibungen beim Nachrichten-Austausch also immer Referenzen ausgetauscht.

6.3.2.2 Parallelisierung

DPDK-NENA erreicht eine Parallelisierung der Datenverarbeitung folgendermaßen. DPDK-NENA teilt die Datenverarbeitung auf verschiedene Prozessor-Kerne auf. Die Anwendungskomponente verwendet einen Prozessor-Kern für die Interaktion mit Anwendungen und den Nachrichtenaustausch in Richtung des Netzes. Die Basiskomponente verwendet einen Prozessor-Kern für den Nachrichtenaustausch über die Netzwerkschnittstelle und den Nachrichtenaustausch in Richtung der Anwendungen. Die verwendeten Prozessor-Kerne werden exklusiv von DPDK-NENA verwendet. Das bedeutet, die Prozessor-Kerne stehen keinen anderen Anwendungen auf dem Endsystem zur Verfügung.

In den folgenden Abschnitten wird näher auf die Anwendungskomponente und die Interaktion mit Anwendungen, auf die Basiskomponente und den Datenaustausch über das Netz sowie auf die Parallelisierung der Nachrichtenverarbeitung eingegangen.

6.4 Anwendungskomponente und Interaktion mit Anwendungen

Die Anwendungskomponente ist für die Interaktion mit Anwendungen zuständig und verwaltet die auf dem Endsystem aktiven Protokoll-Stapel. Die Anwendungskomponente realisiert die Anwendungsschnittstelle in DPDK-NENA und verwaltet Daten- und Kontroll-Ringe für die Kommunikation mit Anwendungen. Die Anwendungskomponente empfängt Nachrichten von Anwendungen und reicht diese an Protokoll-Stapel weiter. Außerdem wird die Anwendungskomponente von Protokoll-Stapeln verwendet, um Nachrichten von Protokoll-Stapeln an Anwendungen weiterzureichen.

Die DPDK-NENA API unterscheidet zwischen Daten- und Kontroll-Nachrichten. Daten-Nachrichten enthalten die eigentlichen Daten, die von Anwendungen ausgetauscht werden. Kontroll-Nachrichten dienen den Anwendungen z.B. zum Auf- und Abbau von

Kommunikationsbeziehungen zu Diensten. Um eine hohe Leistung zu erreichen, verwendet die DPDK-NENA API Ring-Puffer in geteiltem Speicher für den Nachrichtenaustausch mit Anwendungen. Darüber hinaus tauschen Anwendungen mit DPDK-NENA lediglich Referenzen auf Nachrichten aus. Die Nachrichten selbst werden in Nachrichten-Puffern in einem geteilten Speicherbereich abgelegt. Für eine einfache Unterscheidung der Nachrichten von unterschiedlichen Anwendungen und damit sich Anwendungen nicht gegenseitig blockieren, werden für jede Anwendung separate Ring-Puffer verwendet. Damit sich Kontroll- und Daten-Nachrichten nicht gegenseitig blockieren, nutzt zusätzlich jede Anwendung für Kontroll- und für Daten-Nachrichten separate Ring-Puffer. Dies vereinfacht außerdem die Verarbeitung von Kontroll- und Daten-Nachrichten auf unterschiedlichen Prozessor-Kernen.

Für eine flexible Zuordnung von Anwendungen zu Protokoll-Stapeln und eine effiziente Zuordnung von Nachrichten zu Kommunikationsbeziehungen verwendet die Anwendungskomponente zwei Datenstrukturen: Die Liste der Protokoll-Stapel und die Handle Table. In der Liste der Protokoll-Stapel sind alle auf dem Endsystem aktiven Protokoll-Stapel registriert. Sie erlaubt eine einfache Zuordnung von Anwendungen zu Protokoll-Stapeln beim Aufbau von Kommunikationsbeziehungen durch Anwendungen. Die Handle Table wird einerseits dazu verwendet, um von Anwendungen kommende Nachrichten den Protokoll-Stapeln zuzuordnen. Andererseits wird sie dazu benutzt, um Nachrichten von Protokoll-Stapeln den entsprechenden Anwendungen zuzuordnen.

Jede Anwendung benötigt Zugriff auf die geteilten Speicherbereiche, in denen sich die Ring-Puffer und die Nachrichten-Puffer befinden. Diese geteilten Speicherbereiche werden von DPDK-NENA verwaltet und den Anwendungen zugewiesen. Hierfür verwendet DPDK-NENA ein einfaches Registrierungsverfahren. Die Registrierung von Anwendungen erfolgt dabei über einen separaten, globalen Ring-Puffer für Kontroll-Nachrichten, der als Rendezvous-Punkt zwischen DPDK-NENA und Anwendungen dient.

In Abbildung 6.6 werden die Anwendungskomponente und die an der Interaktion mit Anwendungen beteiligten Komponenten dargestellt. Beispielhaft kommunizieren zwei Anwendungen über DPDK-NENA (*Application 1* und *Application 2*). Jede Anwendung verwendet separate Ring-Puffer für Daten- und Kontroll-Nachrichten (*appl1 data* und *appl1 control* bzw. *appl2 data* und *appl2 control*). Der Ring-Puffer *NENA Control* dient bei der Registrierung von Anwendungen als Rendezvous-Punkt zwischen Anwendungen und DPDK-NENA. Die Anwendungskomponente realisiert die DPDK-NENA *API* innerhalb von DPDK-NENA. Sie verwendet als Datenstrukturen die Liste der Protokoll-Stapel Module (*PS Module List*) und die *Handle Table*. Beispielhaft werden zwei Protokoll-Stapel Module dargestellt. Zudem werden die Nachrichten-Puffer (*Message Buffers*) gezeigt, die beim Nachrichten-Austausch zwischen Anwendungen und DPDK-NENA verwendet werden.

Im folgenden wird näher auf (1) das Multiplexen von verschiedenen Anwendungen über DPDK-NENA, (2) auf das Multiplexen verschiedener Kommunikationsbeziehungen einer Anwendung, (3) auf die Zuordnung von Anwendungen zu Protokoll-Stapeln und

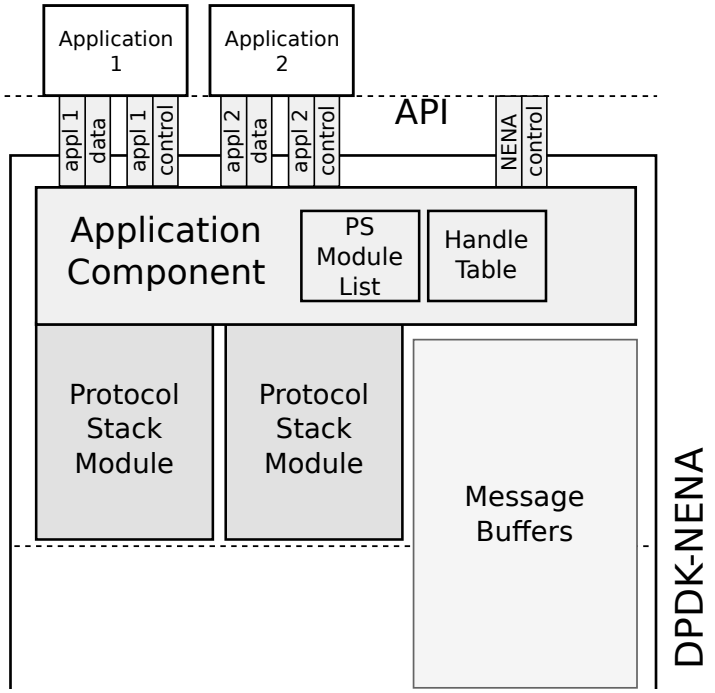

Abbildung 6.6: *An der Interaktion mit Anwendungen beteiligte Komponenten.*

(4) auf die Registrierung von Anwendungen in DPDK-NENA eingegangen. Abschließend wird die DPDK-NENA API zusammengefasst.

6.4.1 Multiplexen von Anwendungen

Über DPDK-NENA müssen gleichzeitig mehrere Anwendungen Nachrichten austauschen. Erhält DPDK-NENA Nachrichten von Anwendungen müssen sie den verschiedenen Anwendungen zugeordnet werden. Für eine einfache Unterscheidung der Kontroll- und Daten-Nachrichten unterschiedlicher Anwendungen verwendet DPDK-NENA separate Ring-Puffer für verschiedene Anwendungen. Hierdurch wird vermieden, dass eine Anwendung die Kommunikation einer anderen Anwendung blockiert. Selbst wenn eine Anwendung ihren Ring-Puffer mit Nachrichten füllt, können andere Anwendungen über ihre eigenen Ring-Puffer Nachrichten mit DPDK-NENA austauschen. Durch die Verwendung separater Ring-Puffer für Kontroll- und Daten-Nachrichten wird außerdem eine Trennung des Kontroll- und Datenverkehrs erreicht und vermieden, dass Kontroll-Nachrichten von Daten-Nachrichten blockiert werden. Auch wenn der Ring-Puffer für Daten-Nachrichten voll mit Referenzen auf Daten-Nachrichten ist, ist ein Austausch von Kontroll-Nachrichten über den separaten Ring-Puffer möglich. Darüber hinaus verläuft der Nachrichtenaustausch über die Ring-Puffer unidirektional. Das heisst, jede Anwendung verwendet einen Ring-Puffer für den Versand von Daten-Nachrichten und einen

Ring-Puffer für den Empfang von Daten-Nachrichten. Ebenso verwendet jede Anwendung einen Ring-Puffer für den Versand von Kontroll-Nachrichten und einen Ring-Puffer für den Empfang von Kontroll-Nachrichten. Dies erlaubt, den Synchronisationsaufwand beim lesenden und schreibenden Zugriff auf die Ring-Puffer zu reduzieren. Da die Ring-Puffer also in Paaren verwendet werden, werden in dieser Arbeit die Ring-Puffer auch als Paare dargestellt, wie z.B. in Abbildung 6.6. Die Verwendung unterschiedlicher Ring-Puffer vereinfacht außerdem die Parallelisierung der Nachrichtenverarbeitung in DPDK-NENA. Durch die verschiedenen Ring-Puffer wird beispielsweise vereinfacht, Kontroll- und Daten-Nachrichten auf unterschiedlichen Prozessor-Kernen zu verarbeiten. Bei der Verarbeitung von Kontroll-Nachrichten auf einem Prozessor-Kern und bei der Verarbeitung von Daten-Nachrichten auf einem anderen Prozessor-Kern können die Nachrichten einfach aus den entsprechenden Ring-Puffern ausgelesen und weiterverarbeitet werden. Zudem wird, dadurch dass über die Ring-Puffer nur Referenzen auf Nachrichten ausgetauscht werden, der Speicheraufwand für die verschiedenen Ring-Puffer gering gehalten und ist vergleichbar mit der Verwendung eines einzelnen, entsprechend großen, Ring-Puffers.

6.4.2 Multiplexen von Kommunikationsbeziehungen

Eine Anwendung kann zur selben Zeit verschiedene Kommunikationsbeziehungen zu unterschiedlichen Diensten unterhalten. Beispielsweise könnte eine Anwendung gleichzeitig eine Datei herunterladen und Text-Nachrichten zwischen Nutzern austauschen. Die Nachrichten der Anwendung müssen den entsprechenden Kommunikationsbeziehungen zugeordnet werden, um sie den entsprechenden Protokoll-Stapeln zuordnen zu können. Für eine effiziente Zuordnung von Nachrichten zu Kommunikationsbeziehungen unterscheidet DPDK-NENA Kommunikationsbeziehungen anhand eines so genannten Handles. Ein Handle stellt eine auf dem Endsystem eindeutige Identifikationsnummer der Kommunikationsbeziehung dar. Die Nachrichten einer Anwendung werden über ein Feld im Nachrichtenkopf, das das Handle der Kommunikationsbeziehung enthält, den verschiedenen Kommunikationsbeziehungen zugeordnet.

6.4.3 Zuordnung von Anwendungen zu Protokoll-Stapeln

Kommunikationsbeziehungen von Anwendungen zu Diensten können über verschiedene Protokoll-Stapel verlaufen. Für einen schnellen Datenaustausch zwischen Anwendungen und dem Netz müssen daher Nachrichten von den Anwendungen schnell den verschiedenen Protokoll-Stapeln zugeordnet werden. Wie zuvor beschrieben, verwendet DPDK-NENA Handles, um Nachrichten Kommunikationsbeziehungen zuzuordnen. Für eine schnelle Zuordnung von Kommunikationsbeziehungen zu Protokoll-Stapeln nutzt DPDK-NENA die Handle Table. Die Handle Table ist eine Hash-Tabelle, die von Protokoll-Stapel Modulen kommende Nachrichten anhand des Handle den Anwendungen und

von Anwendungen kommende Nachrichten anhand des Handle den Protokoll-Stapel Modulen zuordnet.

Für das Anlegen der Zuordnungen verwendet DPDK-NENA ein flexibles Verfahren, das es den Protokoll-Stapel Modulen erlaubt, zu entscheiden, ob sie Nachrichten von Anwendungen verarbeiten. Das Anlegen der Zuordnung einer Kommunikationsbeziehung zu einem Protokoll-Stapel erfolgt beim Aufbau einer Kommunikationsbeziehung. Wenn eine Anwendung eine Kommunikationsbeziehung aufbaut, sucht DPDK-NENA einen geeigneten Protokoll-Stapel für die Kommunikationsbeziehung. Die Anfrage der Anwendung, die die Anwendungskomponente über den Kontroll-Ring der Anwendung erhält, enthält den Namen des Dienstes, auf den die Anwendung zugreifen will. Der Name wird als Parameter für folgenden einfachen Such-Algorithmus verwendet.

```
findPSModule(name):
        for PS-Module in PS-Module-List:
                if PS-Module.canHandle(name) is True:
                        return PS-Module
        return Error
```

Mit Hilfe der Liste der Protokoll-Stapel Module befragt die Anwendungskomponente alle aktiven Protokoll-Stapel Module auf dem Endsystem, ob sie den gewünschten Dienst zur Verfügung stellen können. Hierzu wird die *canHandle()* Funktion jedes Protokoll-Stapel-Moduls in der Liste mit dem Namen als Parameter aufgerufen. Die Protokoll-Stapel-Module überprüfen anhand des Namens, ob sie die Anfrage bearbeiten können. Die genaue Art der Überprüfung ist Modul-spezifisch. Liefert ein Protokoll-Stapel ein positives Ergebnis zurück, wird der Kommunikationsbeziehung ein Handle zugewiesen und das Handle sowohl mit der Anwendung als auch dem Protokoll-Stapel in der Handle Table assoziiert. Abschließend wird der Anwendung das Handle über den Kontroll-Ring zurückgegeben. Nachfolgende Nachrichten der Anwendung, die das Handle enthalten, können direkt über den Eintrag in der Handle Table dem entsprechende Protokoll-Stapel Modul zugeordnet werden.

6.4.4 Registrierung von Anwendungen

Für einen schnellen Nachrichtenaustausch zwischen DPDK-NENA und Anwendungen werden Ring-Puffer und Nachrichten-Puffer in geteilten Speicherbereichen verwendet. DPDK-NENA verwaltet diese Speicherbereiche. Anwendungen, die auf dem Endsystem ausgeführt werden, benötigen Zugriff auf diese Speicherbereiche. Um Ring-Puffer für Anwendungen anzulegen und Anwendungen Zugriff auf die geteilten Speicherbereiche zu geben, verwendet DPDK-NENA das folgende Verfahren zur Registrierung von Anwendungen. Registrierungsnachrichten werden als Kontroll-Nachrichten zwischen Anwendungen und DPDK-NENA über spezielle globale Ring-Puffer, die als Rendez-vous Punkt dienen, ausgetauscht. Die Namen dieser Ring-Puffer sind statisch festgelegt. Hierdurch

kann sich eine neu gestartete Anwendung mit den Ring-Puffern über den festgelegten Namen verbinden und den initialen Informationsaustausch beginnen. Da über die Ring-Puffer nur Referenzen auf Nachrichten, die in einem geteilten Speicherbereich abgelegt werden, ausgetauscht werden, benötigt die Anwendung für den initialen Informationsaustausch Zugriff auf einen solchen Speicherbereich. DPDK-NENA verwendet einen statisch festgelegten Speicherbereich für Registrierungsnachrichten. Dadurch kann sich eine Anwendung beim Start direkt mit diesem Speicherbereich verbinden und die Registrierung mit DPDK-NENA beginnen.

Um den Nachrichtenaustausch über die globalen Ring-Puffer beim Start mehrerer Anwendungen zu regulieren, wird folgender Synchronisationsmechanismus verwendet. Der Speicherbereich für die Registrierungsnachrichten enthält genau einen Nachrichten-Puffer und auf diesen kann nur eine Anwendung gleichzeitig zugreifen. Die Anwendung, die zuerst auf diesen Nachrichten-Puffer zugreift, darf sich bei DPDK-NENA registrieren. DPDK-NENA verwendet den selben Nachrichten-Puffer, um Antworten an die Anwendung zurück zu schicken. Die anderen Anwendungen müssen warten, bis der Nachrichten-Puffer nach der Registrierung von der Anwendung wieder freigegeben wird.

Beim initialen Informationsaustausch über die Ring-Puffer, registriert sich zunächst die Anwendung. Daraufhin werden von der Anwendungskomponente neue Ring-Puffer für die Kontroll- und Daten-Nachrichten der Anwendung angelegt. Die Namen der neuen Ring-Puffer und des geteilten Speicherbereichs für Nachrichten-Puffer werden über die globalen Ring-Puffer zurück an die Anwendung gesendet. Die Anwendung kann sich daraufhin mit den neuen Ring-Puffern und dem Speicherbereich für Nachrichten-Puffer verbinden und über diese weitere Nachrichten mit DPDK-NENA austauschen.

6.4.5 Interaktion über die DPDK-NENA API

Abschließend werden an dieser Stelle die DPDK-NENA API und die Interaktion mit Anwendungen zusammengefasst. Die DPDK-NENA API wird mit Hilfe der Ring-Puffer für Daten- und Kontroll-Nachrichten umgesetzt. Die DPDK-NENA API basiert auf der ursprünglichen NENA-API [71]. Die Befehle der NENA-API werden mit Hilfe von Nachrichten realisiert, die über die Ring-Puffer ausgetauscht werden. Die DPDK-NENA API verwendet die folgenden Nachrichtenformate:

- | **AppMsg** | type | name |
  type: register, unregister, ok, error. *name*: lokal eindeutiger Name der Anwendung.
- | **HandleMsg** | type | name | handle |
  type: get, put, connect, bind, close, ok, error. *name*: Name, mit dem das Handle assoziiert werden soll, z.B. die URI des Dienstes, auf den zugegriffen werden soll. *handle*: Identifikator der Kommunikationsbeziehung.

- | **DataMsg** | handle | data |

  handle: Identifikator der Kommunikationsbeziehung. *data*: Daten, die über die Kommunikationsbeziehung ausgetauscht werden.

Die *AppMsg* Nachrichten werden verwendet, um Anwendungen bei DPDK-NENA zu registrieren oder wieder abzumelden (Registrierung). Die *HandleMsg* und *DataMsg* Nachrichten werden verwendet, um die ursprüngliche NENA API zu realisieren. *HandleMsg* Nachrichten werden verwendet, um Kommunikationsbeziehungen über DPDK NENA auf- und abzubauen. *DataMsg* Nachrichten werden verwendet, um über bestehende Kommunikationsbeziehungen Daten zu empfangen oder zu versenden.

Die Verwendung dieser Nachrichten und der verschiedenen Ring-Puffer in DPDK-NENA wird anhand der in Abbildung 6.7 dargestellten Abläufe näher beschrieben. Es werden (1) die Registrierung einer Anwendung bei DPDK-NENA, (2) das Aufbauen einer Kommunikationsbeziehung über DPDK-NENA und (3) die Nutzung einer Kommunikationsbeziehung zur Übertragung von Daten jeweils ohne das Auftreten von Fehler-Situationen vorgestellt.

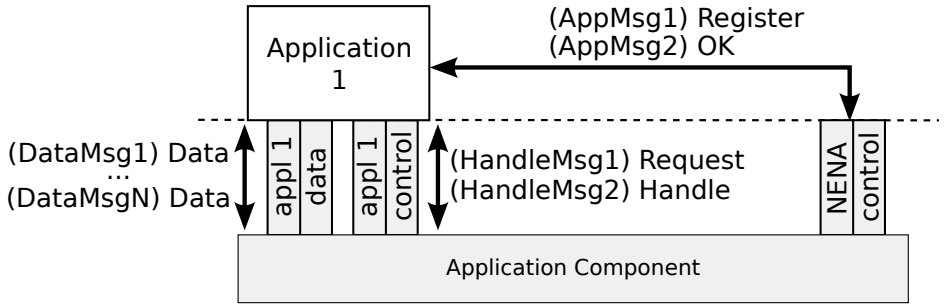

Abbildung 6.7: *Schritte bei der Verwendung der Anwendungsschnittstelle.*

(1) *Registrierung einer Anwendung*: Wenn eine Anwendung auf einem Endsystem gestartet wird, die über DPDK-NENA kommunizieren soll, muss sie zunächst an DPDK-NENA angebunden werden. Hierzu verbindet sich die Anwendung mit den globalen Ring-Puffern von DPDK-NENA (*NENA Control*). Über diese Ring-Puffer sendet die Anwendung eine *AppMsg* Nachricht vom Typ *register*. Diese Nachricht enthält einen lokal eindeutigen Namen der Anwendung. Die Anwendungskomponente in DPDK-NENA nimmt diese Nachricht entgegen. Sie registriert die Anwendung und erstellt zwei neue Paare von Ring-Puffern für die Anwendung: ein Ring-Paar für Daten-Nachrichten (*appl1 data*) und ein Ring-Paar für Kontroll-Nachrichten (*appl1 control*). Diese werden mit dem Namen der Anwendung assoziiert. Daraufhin erstellt die Anwendungskomponente eine *AppMsg* Nachricht vom Typ *OK*, die den Namen der Anwendung enthält, und sendet diese über *NENA Control* zurück an die Anwendung. Die Anwendung liest diese Nachricht von *NENA Control* und verbindet sich mit den beiden neu erstellten Ring-Paaren über ihren Namen. Die Anwendung ist nun in der Lage ein Ring-Paar zu verwenden, um weitere

Kontroll-Nachrichten mit DPDK-NENA auszutauschen. Außerdem kann die Anwendung ein Ring-Paar dazu benutzen, Daten über DPDK-NENA zu empfangen oder zu versenden.

(2) *Aufbauen eines Kommunikationsbeziehung*: Um Daten über DPDK-NENA versenden oder zu empfangen, muss eine Anwendung zunächst eine Kommunikationsbeziehung zu einem Dienst über DPDK-NENA aufbauen. Hierzu erstellt die Anwendung eine *HandleMsg* Nachricht. Das Typ-Feld der Nachricht enthält die Primitive der NENA API, die die Anwendung verwenden will. Zum Beispiel enthält dieses Feld im Falle einer *get()* Anfrage den Wert *get*. Das Feld *Name* enthält den Namen des angeforderten Dienstes. Das Handle-Feld dieser Nachricht wird nicht verwendet und daher mit 0 initialisiert. Die Anwendung sendet diese Nachricht über ihren Ring für Kontroll-Nachrichten an DPDK NENA. Die Anwendungskomponente in DPDK-NENA nimmt die Nachricht entgegen und verarbeitet sie. Sie generiert ein neues Handle *h* für die Anfrage der Anwendung. Für den angeforderten Dienst wählt sie einen Protokoll-Stapel aus und assoziiert das Handle *h* mit dem Protokoll-Stapel und der Anwendung. Anschließend erstellt sie eine neue *HandleMsg* Nachricht vom Typ *ok*, die den von der Anwendung angeforderten Dienst enthält. Außerdem wird in das Handle-Feld das Handle *h* eingetragen. Die Anwendungskomponente sendet die Nachricht an die Anwendung über ihren Ring-Puffer für Kontroll-Nachrichten. Die Anwendung nimmt die Nachricht entgegen und extrahiert das Handle für die neue Kommunikationsbeziehung. Die Anwendung kann nun das Handle verwenden, um Daten über die Kommunikationsbeziehung auszutauschen.

(3) *Nutzung einer Kommunikationsbeziehung*: Nachdem eine Anwendung eine neue Kommunikationsbeziehung aufgebaut hat, kann sie diese nutzen, um Daten auszutauschen. Der Typ der Kommunikationsbeziehung legt fest, ob die Anwendung Daten über die Kommunikationsbeziehung verschicken oder empfangen kann. Um Daten über eine Kommunikationsbeziehung zu verschicken, erstellt die Anwendung eine *DataMsg* Nachricht. In das Handle-Feld dieser Nachricht setzt die Anwendung das Handle der Kommunikationsbeziehung. Außerdem fügt sie die zu versendenden Daten in das Daten-Feld der Nachricht ein. Die Anwendung sendet diese Nachricht an DPDK-NENA über ihren Ring-Puffer für Daten-Nachrichten. Die Anwendungskomponente in DPDK-NENA nimmt die Nachricht vom Ring-Puffer der Anwendung, liest das Handle aus und reicht die Nachricht an den für die Kommunikationsbeziehung verantwortlichen Protokoll-Stapel weiter. Der Protokoll-Stapel ist daraufhin dafür zuständig, die Nachricht weiterzuverarbeiten und zu versenden. Um Daten über eine Kommunikationsbeziehung zu empfangen, liest die Anwendung *DataMsg* Nachrichten von ihrem Ring-Puffer für Daten-Nachrichten. Hierfür müssen die Daten zuvor von dem für die Kommunikationsbeziehung verantwortlichen Protokoll-Stapel verarbeitet und mit dem Handle der Kommunikationsbeziehung versehen werden. Außerdem muss die dabei entstandene *DataMsg* von der Anwendungskomponente in den Ring-Puffer für Daten-Nachrichten der Anwendung gelegt werden. Liest die Anwendung eine *DataMsg* Nachricht von ihrem Ring-Puffer für Daten-Nachrichten, überprüft sie zunächst das Handle in der Nachricht. Die Anwendung

verwendet das Handle, um die Daten der entsprechenden Kommunikationsbeziehung zuzuordnen, und verarbeitet die Daten schließlich weiter.

6.5 Basiskomponente und Datenaustausch über das Netz

Die Basiskomponente ist für die Interaktion mit den Netzwerkschnittstellen verantwortlich und verwaltet die auf dem Endsystem aktiven Module für Virtuelle Verbindungen. Die Basiskomponente liest Pakete von den Netzwerkschnittstellen und leitet diese an Module für Virtuelle Verbindungen weiter. Außerdem wird die Basiskomponente verwendet, um Pakete von Modulen für Virtuelle Verbindungen über das Netzwerk zu versenden.

Um eine hohe Leistung zu erreichen, greift die Basiskomponente direkt auf die Netzwerkschnittstellen des Endsystems zu. Die Basiskomponente ist dadurch in der Lage, Nachrichten aus den Nachrichten-Puffern im geteilten Speicherbereich über Netzwerkschnittstellen zu versenden und Nachrichten über Netzwerkschnittstellen direkt in Nachrichten-Puffer im geteilten Speicherbereich zu empfangen. Dies ermöglicht einen Nachrichtenaustausch zwischen Anwendungen und dem Netz, ohne dass die entsprechenden Daten kopiert werden müssen.

Für eine flexible und effiziente Zuordnung der vom Netz einkommenden Pakete zu Modulen für Virtuelle Verbindungen unterhält die Basiskomponente zwei Datenstrukturen: die Liste der Module für Virtuelle Verbindungen und die Flow Table. In der Liste der Module für Virtuelle Verbindungen werden alle auf dem Endsystem aktiven Module für Virtuelle Verbindungen registriert. Sie erlaubt eine einfache Zuordnung von Paketen zu Modulen für Virtuelle Verbindungen, wenn bisher unbekannte Pakete über das Netz eintreffen. Die Flow Table bildet effizient Pakete auf Module für Virtuelle Verbindungen ab.

In Abbildung 6.8 werden die Basiskomponente und die an dem Datenaustausch mit dem Netz beteiligten Komponenten dargestellt. Beispielhaft verwenden zwei Module für Virtuelle Verbindungen die Basiskomponente, um Daten über das Netz auszutauschen. Die Basiskomponente verwendet einen Ring-Puffer, um Daten von den Modulen für Virtuelle Verbindungen entgegen zu nehmen. Die Basiskomponente verwendet die Liste der Module für Virtuelle Verbindungen und die Flow Table, um die von der Netzwerkschnittstelle (*NIC*) eintreffenden Pakete den Modulen für Virtuelle Verbindungen zuzuordnen. Für den Versand und Empfang von Nachrichten werden die Nachrichten-Puffer im geteilten Speicherbereich verwendet.

Im Folgenden wird näher auf das Versenden von Paketen über die Netzwerkschnittstellen des Endsystems und die Zuordnung von eintreffenden Paketen zu Modulen für Virtuelle Verbindungen eingegangen.

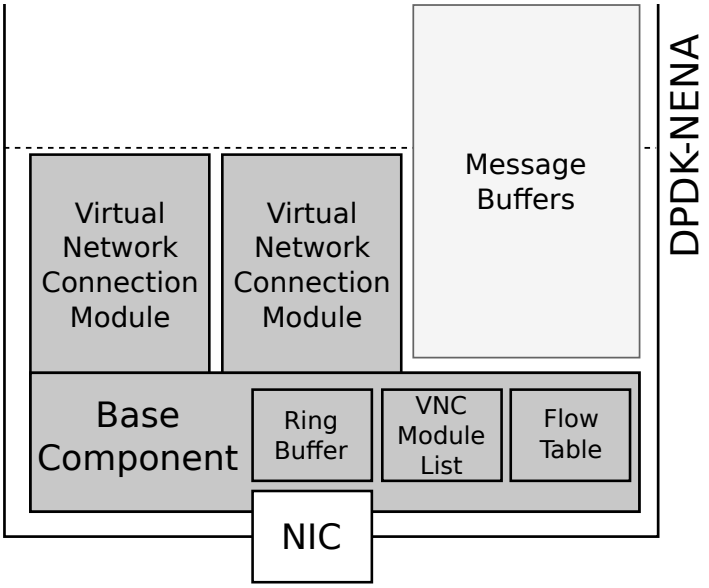

Abbildung 6.8: *An dem Datenaustausch über das Netz beteiligte Komponenten.*

6.5.1 Versenden von Paketen über Netzwerkschnittstellen

Für das schnelle Versenden von Paketen über Netzwerkschnittstellen greift die Basiskomponente direkt auf die Netzwerkschnittstellen des Endsystems zu. Allerdings kann hierdurch nur die Basiskomponente auf die Netzwerkschnittstellen zugreifen. Daher wird in der Basiskomponente ein separater Ring-Puffer pro Netzwerkschnittstelle betrieben. Um Pakete über eine Netzwerkschnittstelle zu versenden, legen Module für Virtuelle Verbindungen daher die Referenzen auf die Nachrichten in den entsprechenden Ring-Puffer in der Basiskomponente. Die Basiskomponente nimmt die Referenzen aus dem Ring-Puffer und versendet die Nachrichten aus den Nachrichten-Puffern im geteilten Speicherbereich über die Netzwerkschnittstelle.

Besitzt das Endsystem mehrere Netzwerkschnittstellen, muss beim Versenden eines Pakets entschieden werden, über welche Schnittstelle das Paket versendet wird. Zum Erreichen von Flexibilität wird diese Entscheidung von den Modulen für Virtuelle Verbindungen getroffen. Hierdurch kann jedes Modul für Virtuelle Verbindungen individuelle Mechanismen für diese Entscheidung implementieren. Damit ein Modul für Virtuelle Verbindungen die Entscheidung, über welche Netzwerkschnittstelle ein Paket versandt wird, fällen kann, benötigt das Modul Wissen über die verfügbaren Netzwerkschnittstellen des Endsystems und über deren Konfiguration. Daher bietet die Basiskomponente die Funktion *getInterfaces()*. Diese Funktion liefert als Rückgabe die Netzwerkschnittstellen und deren Konfiguration. Die Konfiguration beinhaltet die MAC- und IP-Adressen der Netzwerkschnittstellen sowie die Weiterleitungstabelle des Endsystems.

6.5.2 Zuordnung von Paketen zu Virtuellen Verbindungen

Empfängt die Basiskomponente Pakete über Netzwerkschnittstellen, muss sie die Pakete an die Module für Virtuelle Verbindungen weiterleiten, die für die Pakete zuständig sind. Um eine hohe Leistung zu erreichen, muss für jedes eintreffende Paket daher effizient entschieden werden, zu welchem Modul für Virtuelle Verbindungen es gehört.

Für eine hohe Leistung und Flexibilität verwendet die Basiskomponente eine Flow-basierte Zuordnung von Paketen zu Modulen für Virtuelle Verbindungen. Vom Netz eintreffende Pakete werden dabei anhand von Paket-Feldern den Flows zugeordnet und Pakete eines Flows werden einheitlich von der Basiskomponente behandelt. Um flexibel zu sein, wird ein Flow durch möglichst viele Paket-Felder identifiziert. Flows werden daher über das 9-Tupel bestehend aus der Netzwerkschnittstelle, über die das Paket empfangen wurde, der VLAN-ID, der Quell- und Ziel-MAC-Adresse, der IPv4 Quell- und Ziel-Adresse, dem Schicht-4-Protokoll und dem TCP/UDP Quell- und Ziel-Port identifiziert. Da das Festlegen des 9-Tupels ein potentielles Hindernis für die Erweiterbarkeit der Basiskomponente darstellt, könnte alternativ ein n-Tupel z.B. beim Start von DPDK-NENA basierend auf der Konfiguration des Endsystems festgelegt werden. Diese Variante wurde aber im Rahmen dieser Arbeit nicht näher untersucht.

Die Flows werden mit Hilfe der Flow Table den Modulen für Virtuelle Verbindungen zugeordnet. Die Flow Table ist eine Hash-Tabelle und bildet daher effizient Flows auf Module für Virtuelle Verbindungen ab. Als Schlüssel der Hash-Tabelle wird ein Hash über die Paket-Felder, die den Flow identifizieren, verwendet. Falls ein Feld in einem Paket nicht vorhanden ist, wird dabei der Wert als 0 angenommen.

Um Pakete mit Hilfe der Flow Table an Module für Virtuelle Verbindungen weiterleiten zu können, müssen Einträge für die entsprechenden Flows in der Flow Table vorhanden sein. Das Anlegen eines Eintrages in der Flow Table für ein Modul für Virtuelle Verbindung kann nicht beim Laden des entsprechenden Moduls geschehen, da der Ersteller des Moduls bzw. dessen Konfiguration kein Wissen über das Endsystem oder das Netz besitzt. Außerdem wird angenommen, dass das Endsystem nur lokales Wissen über das Netz hat. Es kann daher also auch nicht beim Laden des Moduls für Virtuelle Verbindungen die Konfiguration des Netzes ermittelt und dementsprechend der Flow Table Eintrag generiert werden. Zum Beispiel kann das Endsystem nicht wissen, welche MAC-Adresse die Netzwerkschnittstelle eines Endsystems, z.B. eines Routers, besitzt, von dem Pakete empfangen werden, bevor die entsprechenden Pakete tatsächlich empfangen werden. Die Basiskomponente nutzt daher die Liste der Module für Virtuelle Verbindungen, um die Zuordnung von Paketen zu Modulen für Virtuelle Verbindungen in der Flow Table anzulegen, wenn Pakete bisher unbekannter Flows im Endsystem eintreffen. Trifft ein Paket über das Netz ein, für das es keinen Eintrag in der Flow Table gibt, verwendet die Basiskomponente die folgende Suchfunktion für Module für Virtuelle Verbindungen.

```
findVNCModule(packet):
```

```
        for VNC-Module in VNC-Module-List:
                if VNC-Module.canHandle(packet) is True:
                        return VNC-Module
        return Error
```

Mit Hilfe der Liste der Module für Virtuelle Verbindungen werden alle Module für Virtuelle Verbindungen, die auf dem Endsystem aktiv sind, über die Funktion *canHandle()* mit dem Paket als Parameter abgefragt. Hierdurch kann jedes Modul individuelle Mechanismen implementieren, um zu entscheiden, ob ein Paket zu einer eigenen Virtuellen Verbindung gehört. Wird ein Modul gefunden, das für den Flow zuständig ist, wird ein Eintrag in der Flow Table angelegt und das Paket an das entsprechende Modul weitergereicht. Wird kein Modul gefunden, wird das Paket verworfen. Die Selektion von Modulen ist eine vergleichsweise langsame Operation durch linearen Suchaufwand und gegebenenfalls langsame Operationen in den *canHandle()*-Funktionen der VNC-Module. Allerdings wird die Selektion nur ein Mal pro Virtueller Verbindung durchgeführt, solange sich die beim Flow berücksichtigten Felder der Pakete nicht ändern. Alle weiteren Pakete, die zum selben Flow gehören, werden direkt durch den Eintrag in der Flow Table an das entsprechende Modul für Virtuelle Verbindungen weitergereicht.

Technologien wie Intel FlowDirector (FDIR) erlauben die Filterung des einkommenden Datenverkehrs in Hardware auf der Netzwerkkarte. Die Netzwerkkarte kann, mit Hilfe von Filter-Regeln, empfangene Pakete verschiedenen Ring-Puffern der Netzwerkkarte zuordnen. Durch das Definieren von Filter-Regeln und das Zuordnen von Ring-Puffern zu verschiedenen Modulen für Virtuelle Verbindungen zur Laufzeit könnte die Leistung der Flow Table weiter optimiert werden. Die Verwendung von zusätzlicher Hardware-Unterstützung wurde allerdings in dieser Arbeit nicht weiter verfolgt.

6.6 Parallelisierung der Nachrichtenverarbeitung

Zum Erreichen einer hohen Leistung parallelisiert DPDK-NENA die Nachrichtenverarbeitung. DPDK-NENA verwendet einen Prozessor-Kern für die Basiskomponente und einen Prozessor-Kern für die Anwendungskomponente. Die Basiskomponente liest mit Hilfe eines Prozessor-Kerns Pakete von der Netzwerkschnittstelle und ruft Funktionen aller Komponenten auf dem Weg zur Anwendung auf. Die Basiskomponente legt dann jedes Paket in den Puffer der entsprechenden Anwendung. Die Anwendungskomponente liest mit Hilfe eines anderen Prozessor-Kerns Nachrichten vom Ring-Puffer einer Anwendung und ruft die Funktionen aller Komponenten auf dem Weg zum Netz auf. Die Anwendungskomponente legt dann die resultierenden Pakete in einen Ring-Puffer für ausgehende Pakete. Die Basiskomponente liest mit ihrem Prozessor-Kern die Pakete aus dem Ring-Puffer und versendet schließlich die Nachrichten über die Netzwerkschnittstelle.

In diesem Abschnitt wird die Nachrichtenverarbeitung in DPDK-NENA und die Verwendung von Prozessor-Kernen vorgestellt. Im Folgenden werden der Versand von Nachrichten von Anwendungen über das Netz und die Zustellung von Nachrichten aus dem Netz an Anwendungen näher betrachtet.

6.6.1 Versand von Nachrichten von Anwendungen über das Netz

Die Anwendungskomponente in DPDK-NENA verwendet einen Prozessor-Kern, um Nachrichten von Anwendungen abzurufen und weiterzuverarbeiten. Weitere Prozessor-Kerne können der Anwendungskomponente einfach hinzugefügt werden. Sie werden den Ring-Puffern der Anwendungen zugewiesen und führen die selben hier beschriebenen Schritte durch.

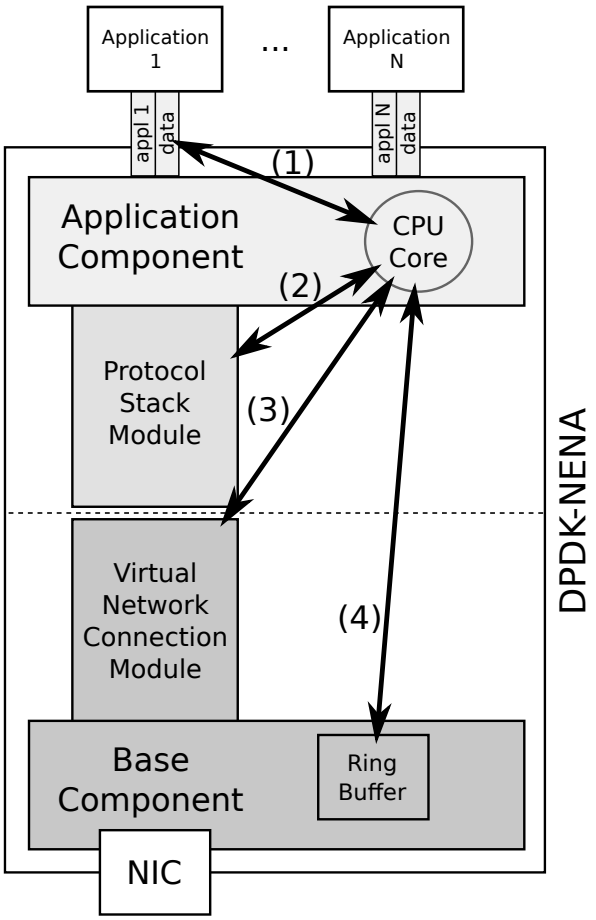

Abbildung 6.9: *Verarbeitung von Nachrichten von Anwendungen durch DPDK-NENA.*

Abbildung 6.9 stellt die Verarbeitung von Nachrichten von Anwendungen durch DPDK-NENA dar. Die Pfeile repräsentieren das Weiterreichen der Nachrichten bzw. die Verarbeitungsschritte in den Komponenten von DPDK-NENA. Die Anwendungskomponente

arbeitet mit einem Prozessor-Kern kontinuierlich die Schritte 1 bis 4 der Reihenfolge nach ab. (1) Im ersten Schritt werden bis zu *B* Pakete aus einem Sende-Puffer einer Anwendung genommen. Hierbei werden die Sende-Puffer der Anwendungen nach dem Round-Robin-Prinzip abgearbeitet. Das heisst, jedes mal, wenn Schritt 1 ausgeführt wird, werden Pakete aus einem anderen Puffer genommen. Nach dem letzten Puffer wird wieder der erste abgearbeitet. (2) Im zweiten Schritt wird für jedes Paket mit Hilfe der Handle Table der dazugehörige Protokoll-Stapel ausgewählt. Jedes Paket wird mit Hilfe des ausgewählten Protokoll-Stapels weiterverarbeitet. Hierzu wird die *processFromApplication* Funktion mit dem entsprechenden Paket als Parameter verwendet. Die Funktion liefert als Rückgabewert das Modul für Virtuelle Verbindungen, in dem das Paket weiterverarbeitet werden soll. (3) Im dritten Schritt wird jedes Paket an das dazugehörige Modul für Virtuelle Verbindungen weitergegeben. Hierzu wird die *processFromProtocolStack* Funktion des Moduls für Virtuelle Verbindungen mit dem entsprechenden Paket als Parameter von der Anwendungskomponente aufgerufen. Die Funktion liefert als Rückgabewert den Ring-Puffer für die Netzwerkschnittstelle über die das Paket versendet werden soll. (4) Im letzten Schritt wird jedes Paket in den Ring-Puffer der Netzwerkschnittstelle gelegt, über die es versendet werden soll.

Die Rückgabe-Werte der aufgerufenen Funktionen der Protokoll-Stapel Module und Module für Virtuelle Verbindungen geben an, ob Nachrichten mit dem aktuellen Prozessor-Kern weiterverarbeitet werden sollen oder nicht. Falls eine Nachricht nicht weiterverarbeitet werden soll, muss die Nachricht von der entsprechender Komponente selbst wieder freigegeben werden.

Für das Versenden der Nachrichten über die Netzwerkschnittstelle wird die Basiskomponente verwendet, die einen anderen Prozessor-Kern für die Nachrichtenverarbeitung nutzt.

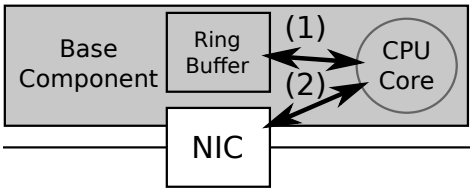

Abbildung 6.10: *Versand von Paketen durch die Basiskomponente.*

Abbildung 6.10 zeigt einen Ausschnitt aus DPDK-NENA und stellt die Nachrichtenverarbeitung beim Versenden von Paketen, die von Anwendungen aus kommen, über eine Netzwerkschnittstelle dar. Die Pfeile repräsentieren die einzelnen Schritte der Nachrichtenverarbeitung. Die Basiskomponente arbeitet mit einem Prozessor-Kern kontinuierlich die Schritte 1 bis 2 der Reihe nach ab. (1) Im ersten Schritt werden bis zu *B* Pakete aus dem Ring-Puffer der Netzwerkschnittstelle ausgelesen. (2) Im zweiten Schritt werden alle ausgelesenen Pakete über die Netzwerkschnittstelle versandt.

6.6.2 Zustellung von Nachrichten aus dem Netz an Anwendungen

Die Basiskomponente nutzt einen Prozessor-Kern pro Netzwerkschnittstelle, um Nachrichten über das Netz zu empfangen, in DPDK-NENA weiterzuverarbeiten und an Anwendungen weiterzureichen. Darüber hinaus können weitere Prozessor-Kerne einfach einer Netzwerkschnittstelle hinzugefügt werden, sofern diese mehrere Ring-Puffer unterstützt. Die Prozessor-Kerne werden den Ring-Puffern der Netzwerkschnittstelle zugewiesen und führen die selben hier beschriebenen Schritte durch.

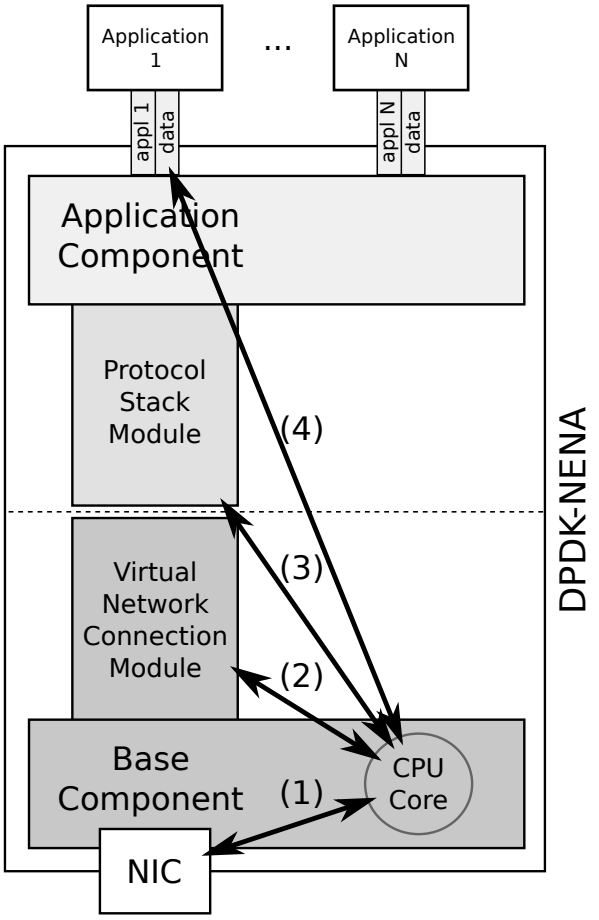

Abbildung 6.11: *Verarbeitung von Nachrichten vom Netz durch DPDK-NENA.*

Abbildung 6.11 stellt die Nachrichtenverarbeitung beim Empfang von Nachrichten vom Netz dar. Die Pfeile repräsentieren die einzelnen Schritte der Nachrichtenverarbeitung. Die Basiskomponente arbeitet mit einem Prozessor-Kern kontinuierlich die Schritte 1 bis 4 der Reihenfolge nach ab. (1) Im ersten Schritt werden bis zu B Pakete von der Basiskomponente über die Netzwerkschnittstelle empfangen. Dabei erhält die Basiskomponente die Referenzen auf die Pakete in den Nachrichten-Puffern. Anschließend wird jedes einzelne der empfangenen Pakete in den folgenden Schritten weiterverarbeitet. Hat jedes

Paket die Schritte 2 bis 4 durchlaufen, wird wieder bei Schritt 1 mit dem Auslesen von bis zu *B* Paketen begonnen. (2) In Schritt 2 wird für jedes Paket mit Hilfe der Flow Table ermittelt, zu welchem Modul für Virtuelle Verbindungen es gehört. Jedes Paket wird dann durch das entsprechende Modul für Virtuelle Verbindungen weiterverarbeitet. Hierzu ruft die Basiskomponente die *processFromNetwork* Funktion des Moduls für Virtuelle Verbindungen mit dem entsprechenden Paket als Parameter auf. Der Rückgabewert der Funktion liefert das Protokoll-Stapel Modul, in dem das Paket weiterverarbeitet werden soll. (3) In Schritt 3 wird jedes Paket durch das entsprechende Protokoll-Stapel Modul weiterverarbeitet. Hierzu ruft die Basiskomponente die *processFromNetwork* Funktion des Protokoll-Stapel Moduls mit dem entsprechenden Paket als Parameter auf. (4) Im letzten Schritt wird mit Hilfe der Handle Table die Anwendung bzw. der Ring-Puffer der Anwendung ermittelt, an die das Paket ausgeliefert werden soll. Jedes Paket wird abschließend in den Ring-Puffer der entsprechende Anwendung eingefügt.

6.7 Implementierung

Um DPDK-NENA evaluieren zu können, wurde ein Prototyp von DPDK-NENA mit Hilfe des Data Plane Development Kits (DPDK) implementiert. In diesem Abschnitt werden Implementierungsdetails des Prototypen vorgestellt.

Randbedingungen

Im Rahmen dieser Arbeit standen zwei identische Test-Rechner zur Verfügung. Jeder dieser Rechner besitzt einen Intel i7-4770 Prozessor mit acht logischen (*Hyper-Threading*) Prozessor-Kerne und 3.4 GHz Taktfrequenz. Die logischen Prozessor-Kerne werden auf vier physikalischen Prozessor-Kernen betrieben. Jedem Rechner stehen 16 GB Arbeitsspeicher und eine Intel x520-DA2 Netzwerkkarte zur Verfügung. Die Netzwerkkarte bietet zwei Netzwerkschnittstellen mit einer maximalen Datenrate von 10 Gigabit/s.

Auf beiden Rechnern wird das Betriebssystem Ubuntu Linux 14.04 mit Linux Kernel Version 3.13.0 betrieben. Für die Implementierung von DPDK-NENA stand DPDK in der Version 1.7.0 zur Verfügung. DPDK basiert auf der Programmiersprache *C*. Daher wurden DPDK-NENA und die Testanwendungen in der Programmiersprache *C* implementiert.

Verwendung von Prozessor-Kernen

Wegen der Verwendung von *Hyper-Threading* im Prozessor, können von den acht logischen Kernen für eine maximale Leistung tatsächlich nur vier Prozessor-Kerne verwendet werden. Diese verwendeten logischen Prozessor-Kerne wurden so gewählt, dass jeder logische Prozessor-Kern auf einem anderen physikalischen Prozessor-Kern betrieben wird. Da das Betriebssystem und die Anwendungen jeweils mindestens einen Prozessor-Kern

benötigen, bleiben somit nur zwei Prozessor-Kerne für DPDK-NENA übrig. DPDK-NENA verwendet einen Prozessor-Kern für die Basiskomponente und einen Prozessor-Kern für die Anwendungskomponente. Der Prozessor-Kern in der Anwendungskomponente liest Nachrichten aus den Ring-Puffern der Anwendungen, verarbeitet sie in den Protokoll-Stapel-Modulen und den Modulen für Virtuelle Verbindungen weiter und übergibt sie an die Basiskomponente. Der Prozessor-Kern in der Basiskomponente versendet die von der Anwendungskomponente erhaltenen Nachrichten über die Netzwerkschnittstelle. Außerdem empfängt die Basiskomponente mit diesem Prozessor-Kern Nachrichten vom Netz, verarbeitet sie in Modulen für Virtuelle Verbindungen und Protokoll-Stapel-Modulen weiter und übergibt sie den Anwendungen.

Anwendungskomponente

Die Anwendungskomponente verwendet DPDK Ring-Puffer für die Kommunikation mit Anwendungen. Für jede Anwendung werden zwei Ring-Puffer für Daten-Nachrichten und zwei Ring-Puffer für Kontroll-Nachrichten verwendet. Dabei wird jeweils ein Ring-Puffer für das Senden von Nachrichten an eine Anwendung und ein Ring-Puffer für das Empfangen von Nachrichten von einer Anwendung verwendet. Die Anwendungskomponente fragt die Ring-Puffer der Anwendungen nach dem Round-Robin Prinzip ab. Dabei werden am Stück bis zu 32 Nachrichten von einer Anwendung gelesen und in DPDK-NENA weiterverarbeitet.

Basiskomponente

Die Basiskomponente verwendet DPDK Ring-Puffer für die Implementierung der Ring-Puffer, die für ausgehende Pakete verwendet werden. Jeder Netzwerkschnittstelle wird ein solcher Ring-Puffer zugeordnet. Die Basiskomponente nimmt Pakete aus diesen Ring-Puffern und versendet sie über die entsprechende Netzwerkschnittstellen. Mit einem Prozessor-Kern für die Basiskomponente wird nur eine der auf dem Rechner verfügbaren Netzwerkschnittstellen verwendet.

Die Flow Table verwendet die CRC32 Hash-Funktion von DPDK und ein Array für die in der Hash-Tabelle abgelegten Informationen. Für den Schlüssel wird der Hash über das 9-Tupel bestehend aus der Nummer der Netzwerkschnittstelle, der MAC Quell- und Ziel-Adresse, der VLAN-ID, der IP Quell- und Ziel-Adresse, dem Schicht-4-Protokoll, dem Schicht-4 Quell- und Ziel-Port berechnet. Sind Felder in einem eintreffenden Paket nicht vorhanden, werden sie bei der Berechnung des Hashes auf den Wert 0 gesetzt. Die Flow Table unterstützt nur IPv4.

Ring-Puffer

DPDK-NENA verwendet DPDK Ring-Puffer für die Implementierung der Ring-Puffer für die Kommunikation mit den Anwendungen und für das Senden über Netzwerkschnittstellen in der Basiskomponente. Diese Ring-Puffer können in verschiedenen Varianten verwendet werden. Mögliche Varianten sind *single consumer*, *single producer*, *multiple consumers* und *multiple producers* und geben an, ob mehrere Prozessor-Kerne auf einen Ring-Puffer lesend oder schreibend zugreifen. Durch eine Einschränkung der Zugriffsmöglichkeiten auf die Ring-Puffer kann der Synchronisationsaufwand reduziert und die Leistung gesteigert werden. Für die Ring-Puffer in DPDK-NENA werden folgende Varianten verwendet.

Ring-Puffer für das Versenden von Nachrichten an eine Anwendung: Nur die Anwendung selbst liest von dem Ring-Puffer, den DPDK-NENA benutzt, um Nachrichten an die Anwendung zu versenden. Da in der Evaluation DPDK-NENA auch nur einen Prozessor-Kern für den Zugriff auf diese Ring-Puffer verwendet, werden sie als *single consumer* und *single producer* Variante betrieben.

Ring-Puffer für das Empfangen von Nachrichten von einer Anwendung: Nur die Anwendung selbst schreibt auf den Ring-Puffer, den DPDK-NENA benutzt, um Nachrichten von der Anwendung zu empfangen. Da die Anwendungskomponente nur einen Prozessor-Kern verwendet, liest auch nur ein Prozessor-Kern Nachrichten von diesem Ring-Puffer. Daher wird dieser Ring-Puffer als *single consumer* und *single producer* Variante betrieben.

Ring-Puffer für das Versenden von Nachrichten über eine Netzwerkschnittstelle: Nur die Basiskomponente liest Nachrichten von dem Ring-Puffer, in den Nachrichten zum Versenden über eine Netzwerkschnittstelle geschrieben werden. Da die Basiskomponente nur einen Prozessor-Kern verwendet, liest somit auch nur ein Prozessor-Kern Nachrichten von dem Ring-Puffer. Allerdings wurde nicht ausgeschlossen, dass mehrere Prozessor-Kerne Nachrichten in diesen Ring-Puffer schreiben. Daher wird dieser Ring-Puffer als *single consumer* und *multiple producers* Variante betrieben.

6.8 Evaluierung

In diesem Abschnitt wird die Evaluierung von DPDK-NENA vorgestellt. Anhand von Experimenten wird die Funktionsfähigkeit von DPDK-NENA demonstriert und in Leistungsmessungen werden die erreichbaren Paket- und Datenraten ermittelt. Dabei wird gezeigt, dass DPDK-NENA in der Lage ist, hohe Paketraten zu erreichen und 10 Gigabit/s Netzwerkschnittstellen auszulasten.

6.8.1 Versuchsaufbau

DPDK-NENA wird mit Hilfe des Versuchsaufbaus evaluiert, der in Abbildung 6.12 dargestellt wird. Zwei Endsysteme, Knoten 1 und Knoten 2, sind direkt über eine 10 Gigabit/s

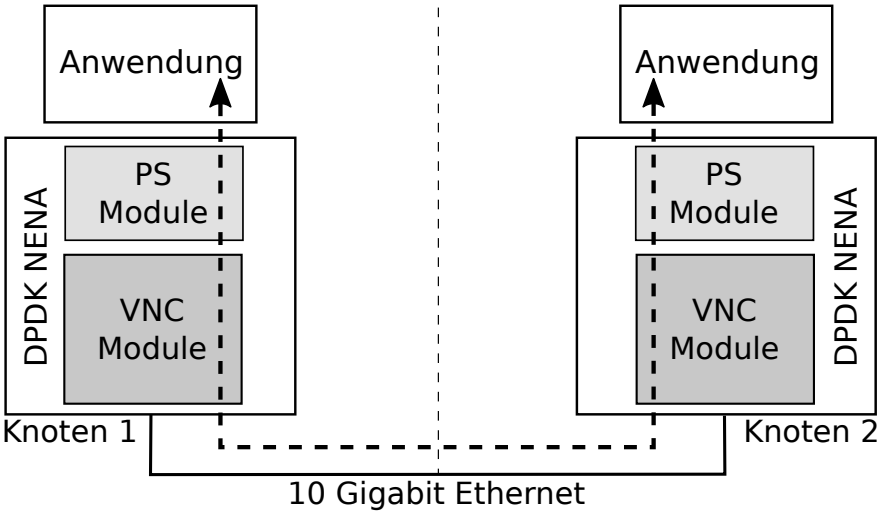

Abbildung 6.12: *Überblick über den Versuchsaufbau.*

Netzwerkschnittstelle miteinander verbunden. Beide Endsysteme verwenden einen Intel i7-4770 Prozessor mit vier physikalischen Prozessor-Kernen, 16 GB RAM und eine Intel x520-DA2 Netzwerkkarte. Das Betriebssystem beider Endsysteme ist Ubuntu Linux 14.04 mit dem Linux Kernel der Version 3.13.0 und DPDK Version 1.7.0. Auf beiden Endsystemen wird DPDK-NENA ausgeführt. In DPDK-NENA werden ein Protokoll-Stapel-Modul und ein Modul für Virtuelle Verbindungen betrieben. Auf Knoten 1 und auf Knoten 2 wird eine Anwendung ausgeführt.

Jedem Endsystem stehen vier Prozessor-Kerne, Prozessor-Kern 1-4, zur Verfügung. Der Prozessor-Kern 1 wird für das Betriebssystem reserviert. DPDK-NENA werden die Prozessor-Kerne 2 und 3 zugeordnet. DPDK-NENA verwendet Prozessor-Kern 2 für die Anwendungskomponente und Prozessor-Kern 3 für die Basiskomponente. Der Prozessor-Kern 4 wird für die Anwendung auf dem Endsystem verwendet. Den DPDK Anwendungen auf einem Endsystem stehen maximal acht Gigabyte *huge page* Speicher der insgesamt vorhandenen 16 Gigabyte Speicher zur Verfügung.

DPDK-NENA verwendet auf beiden Endsystemen ein Protokoll-Stapel-Modul und ein Modul für Virtuelle Verbindungen. Die Protokolle in diesen Modulen hängen vom untersuchten Anwendungsfall ab (siehe unten). Außerdem sind die Protokolle so konfiguriert, dass sie permanent Pakete austauschen können. Sämtliche dafür benötigten Informationen werden als bekannt angenommen. Es werden also keine zusätzlichen Kontroll-Nachrichten, z.B. zum Austausch der Handles oder Aufbauen von Verbindungen, verwendet.

6.8.2 Anwendungsfälle

DPDK-NENA wird mit Hilfe von zwei Anwendungsfällen evaluiert. Diese unterscheiden sich durch die verwendeten Protokoll-Stapel-Module und Module für Virtuelle Verbindungen. Im ersten Anwendungsfall, der *Basisfall*, werden in den Modulen die Protokolle betrieben, die minimal für einen Datenaustausch zwischen den beiden Endsystemen benötigt werden. Dadurch ist der durch die Paketverarbeitung verursachte Aufwand minimal und die maximale Leistung von DPDK-NENA kann ermittelt werden. Im zweiten Anwendungsfall, *UDP/Custom*, werden in den Modulen Protokolle verwendet, die für einen Datenaustausch über das Internet verwendet werden könnten. Dieser Anwendungsfall erlaubt durch die im Internet gängigen Protokolle und aufwändigere Paketverarbeitung in den Modulen eine Untersuchung der Leistung unter realistischeren Bedingungen als der Basisfall.

6.8.2.1 Basisfall

Im Basisfall werden die Nachrichten der Anwendungen direkt in Ethernet-Paketen übertragen. Der verwendete Protokoll-Stapel reicht die von der Anwendung erhaltenen Nachrichten unverändert an das Modul für Virtuelle Verbindungen weiter. Dabei bleibt auch der Nachrichtenkopf, der das Handle der Kommunikationsbeziehung angibt, erhalten. Das Modul für Virtuelle Verbindungen fügt den Nachrichten einen Ethernet-Paketkopf hinzu und versendet sie über die Netzwerkschnittstelle. Empfängt dieses Modul Nachrichten über die Netzwerkschnittstelle, entfernt es den Ethernet-Paketkopf und reicht sie an den Protokoll-Stapel weiter. Dieser reicht die Nachricht unverändert an die Anwendung weiter. Hierfür sind die Endsysteme in diesem Anwendungsfall so konfiguriert, dass die Kommunikationsbeziehung zwischen den Anwendungen auf beiden Endsystemen durch das selbe Handle identifiziert wird.

6.8.2.2 UDP/Custom

Im Anwendungsfall UDP/Custom werden die Nachrichten der Anwendungen über einen UDP-Tunnel und ein einfaches Dienst-spezifisches Protokoll übertragen. Das Dienst-spezifische Protokoll im Protokoll-Stapel erlaubt eine unzuverlässige Übertragung von Daten und benutzt den folgenden Nachrichtenkopf. Ein Byte gibt den Typ der Nachricht an. Vier Bytes werden für die Übertragung des Handles verwendet, das die Kommunikationsbeziehung auf dem sendenden Endsystem identifiziert. Vier Bytes geben das Handle an, das die Kommunikationsbeziehung auf dem empfangenden Endsystem identifiziert. Außerdem werden zwei Bytes verwendet, um die Länge der Nutzdaten anzugeben. Nachrichten des Protokoll-Stapel-Moduls werden über einen UDP-Tunnel ausgetauscht. Daher verwendet das Modul für Virtuelle Verbindungen die Protokolle Ethernet, IPv4 und UDP. Das Modul berechnet statische Felder der Paketköpfe bei der Initialisierung vor.

So ist es z.B. nicht mehr nötig, zur Laufzeit IP- und MAC-Adressen zu ermitteln oder die Prüfsumme des IP-Paketkopfes zu berechnen.

Der verwendete Protokoll-Stapel verarbeitet die von der Anwendung erhaltenen Nachrichten. Dabei entfernt er den Nachrichtenkopf der DPDK-NENA API und fügt den eigenen Nachrichtenkopf hinzu. Anschließend reicht er die Nachricht an das Modul für Virtuelle Verbindungen weiter. Das Modul für Virtuelle Verbindungen fügt den Nachrichten Ethernet-, IP-, und UDP-Paketköpfe hinzu und versendet sie über die Netzwerkschnittstelle. Empfängt dieses Modul Nachrichten über die Netzwerkschnittstelle, entfernt es die Ethernet-, IP, und UDP-Paketköpfe und reicht sie an den Protokoll-Stapel weiter. Dieser entfernt den eigenen Nachrichtenkopf, fügt den Nachrichtenkopf der DPDK-NENA API hinzu und reicht die Nachricht an die Anwendung weiter.

6.8.3 Anwendung

Auf den Endsystemen wird eine Anwendung ausgeführt, die sowohl als Datenquelle als auch als Datensenke in den Experimenten dient. Sie sendet und empfängt Nachrichten über DPDK-NENA mit der maximal möglichen Geschwindigkeit und misst dabei die erzielten Paketraten.

Die Nachrichtenverarbeitung beim Versenden von Nachrichten verläuft wie vereinfacht in folgendem Pseudocode dargestellt.

```
sendPacket():
        pkt = getMsgBuffer()
        addDataMsgHeader(pkt, handle)
        addData(pkt, dataSize)
        send(pkt)
```

In einer Endlosschleife werden die folgenden Operationen durchgeführt. Die Anwendung greift auf den gemeinsamen Speicher für Nachrichten zu. In den Speicher werden die Paketköpfe der DPDK-NENA API für Daten-Nachrichten geschrieben. Anschließend wird die Größe der Nachricht um die gewünschte Länge der Nutzdaten verlängert. Zuletzt wird die Nachricht über DPDK-NENA versandt. Zusätzlich zu den hier gezeigten Operationen misst die Datenquelle die Paketraten beim Senden.

Die Nachrichtenverarbeitung beim Empfangen von Nachrichten verläuft wie vereinfacht in folgendem Pseudocode dargestellt.

```
receivePacket():
        pkt = receive()
        freeMsgBuffer(pkt)
```

In einer Endlosschleife werden die folgenden Operationen durchgeführt. Die Anwendung empfängt von DPDK-NENA eine Nachricht und gibt den Speicher der Nachricht wieder für DPDK-NENA frei. Zusätzlich zu den hier gezeigten Operationen misst die Anwendung die Paketrate beim Empfangen.

Diese Anwendung erlaubt es, die Paketraten zu messen, die beim Datenaustausch zwischen Anwendungen über DPDK-NENA maximal möglich sind. Basierend auf den in den Anwendungen gemessenen Paketraten und der Größe der über das Netz übertragenen Nachrichten können außerdem die möglichen Datenraten errechnet werden. Somit können die Anwendungen genutzt werden, um die maximale Leistung aller bei der Kommunikation beteiligten Komponenten—Anwendungen, DPDK-NENA und Netz—zu evaluieren.

6.8.4 Vergleich mit NENA

Bei der Evaluierung ist ein Vergleich mit dem ursprünglichen NENA Prototypen interessant. Allerdings wurde in [73] gezeigt, dass der ursprüngliche Prototyp von NENA nur in der Lage ist, eine Datenrate von 8 Megabyte/s bei 500 Byte langen Paketen zwischen zwei NENA-Instanzen auf dem selben Endsystem zu erreichen. Außerdem sieht der ursprüngliche Prototyp von NENA keine Realisierung der Virtuellen Verbindungen innerhalb von NENA vor. Daher müssten Module für Virtuelle Verbindungen in den Protokoll-Stapeln für DsNs integriert werden. Aus diesem Grund wurde für einen Vergleich in der Evaluierung eine alternative Version von DPDK-NENA verwendet. In dieser Version wurden die DPDK-basierten Sende- und Empfangsfunktionen in der Basiskomponente durch AF_PACKET[1] Socket Versionen ersetzt. Hierdurch werden die Pakete, hier Ethernet-Pakete, über das Betriebssystem ausgetauscht. Außerdem erfolgt keine Burst-Verarbeitung von Nachrichten beim Datenaustausch über die Netzwerkschnittstelle. Pakete werden einzeln über die Netzwerkschnittstelle empfangen und versandt. Allerdings werden ansonsten alle weiteren Optimierungen von DPDK-NENA, wie der geteilte Speicher für die Kommunikation mit den Anwendungen und die Aufteilung auf mehrere Prozessor-Kerne, verwendet. Somit wird in der Evaluierung DPDK-NENA quasi mit einer optimierten Version des ursprünglichen NENA-Rahmenwerkes verglichen. Darüber hinaus wird gezeigt, welchen Einfluss die Optimierung des Datenaustausches mit dem Netz auf die Leistungsfähigkeit von DPDK-NENA hat.

6.8.5 Leistung

Für die Evaluierung der Leistung von DPDK-NENA wurde der weiter oben beschriebene Versuchsaufbau verwendet und es wurden die beiden Anwendungsfälle *Basisfall* und *UDP/Custom* untersucht. Für jeden Anwendungsfall wurde die Leistung einer unidirektionalen und einer bidirektionalen Übertragung evaluiert. Hierfür wurden die folgenden Leistungsmessungen durchgeführt:

[1] Ein AF_PACKET Socket ist eine Socket Variante, die das Senden und Empfangen von Ethernet-Paketen ermöglicht

- *Leistung unidirektional*:
  - *Sendeleistung*: Um die Sendeleistung in einer Anwendung zu evaluieren, generiert die Anwendung auf Knoten 1 so schnell wie möglich Nachrichten und versendet sie über DPDK-NENA. Die Nachrichten der Anwendung werden im Protokoll-Stapel-Modul und im Modul für Virtuelle Verbindungen verarbeitet und über die Netzwerkschnittstelle an Knoten 2 versandt. Die Anwendung misst die beim Versand erzielte Paketrate.
  - *Empfangsleistung*: Um die Empfangsleistung in einer Anwendung zu evaluieren, generiert die Anwendung auf Knoten 1 wieder einen konstanten Strom an Nachrichten mit maximal möglicher Geschwindigkeit und sendet sie über DPDK-NENA an Knoten 2. Hierzu werden die Nachrichten der Anwendung im Protokoll-Stapel-Modul und im Modul für Virtuelle Verbindungen verarbeitet und über die Netzwerkschnittstelle versandt. DPDK-NENA auf Knoten 2 empfängt die Nachrichten über das Netzwerk und verarbeitet sie im Modul für Virtuelle Verbindungen und dem Protokoll-Stapel-Modul weiter. DPDK-NENA reicht die Nachrichten an die Anwendung weiter, welche diese empfängt und die erzielte Paketrate misst.

- *Leistung bidirektional*:
  - *Sende- und Empfangsleistung*: Um die Leistung in einer Anwendung bei gleichzeitigem Senden und Empfangen von Daten zu evaluieren, versendet die Anwendung auf Knoten 1 wieder Nachrichten so schnell wie möglich an Knoten 2. Auf Knoten 2 werden die Nachrichten verarbeitet und an die dortige Anwendung weitergereicht. Diese misst die einkommende Paketrate. Zusätzlich versendet diese Anwendung selbst auch einen konstanten Strom von Nachrichten mit maximal möglicher Geschwindigkeit an Knoten 1. Dabei misst sie die beim Senden erreichte Paketrate.

In den Experimenten wurde die Menge der Nutzdaten, die zwischen den Anwendungen in jeder Nachricht ausgetauscht wird, variiert. Es wurden Nutzdaten der Länge 0, 10, 25, 50, 100, 200, 400, 800 und 1400 Byte untersucht. Bei 0 Byte Nutzdaten bestehen die Nachrichten der Anwendung nur aus dem Nachrichtenkopf der DPDK-NENA API. Für jede Nutzdatenlänge wurden zehn Versuchsläufe durchgeführt, in denen die Zeit gemessen wurde, die für das Senden bzw. den Empfang von einer Milliarde Nachrichten benötigt wird. Dadurch wurde für jeden Versuchslauf die Paketrate bestimmt. Bei der Untersuchung der Socket-basierten Version von NENA wurden in jedem Versuchslauf 100 Millionen Nachrichten übertragen, da dort die erzielten Paketraten geringer waren. Mit Hilfe der gemessenen Paketraten, der Nutzdatenlänge und des von den Protokollen des Protokoll-Stapel-Moduls und des Moduls für Virtuelle Verbindungen

verursachten Overhead wurden die erzielten Datenraten bei der Übertragung über die Netzwerkschnittstelle ermittelt. Der Overhead in den beiden Anwendungsfällen ist der folgende:

- Im *Basisfall* werden die Nachrichten der Anwendungen direkt über Ethernet übertragen. Die über die Netzwerkschnittstelle ausgetauschten Nachrichten enthalten daher den Nachrichtenkopf der DPDK-NENA API (4 Byte) und einen Ethernet-Paketkopf (18 Byte). Der bei der Übertragung der Nutzdaten über das Netzwerk verursachte Overhead beträgt damit 42 Byte (inkl. des 10 Gigabit Ethernet-spezifischen *Interframe Gap* und der Präambel (zusammen 20 Byte)). Im Falle von 0, 10 und 25 Byte Nutzdaten werden die Nachrichten bei dem Versand durch die Netzwerkschnittstelle auf die Minimallänge von 10 Gigabit/s Ethernet verlängert (84 Byte).
- Im Anwendungsfall *UDP/Custom* werden die Nachrichten der Anwendung über ein unzuverlässiges Protokoll und einen UDP-Tunnel übertragen. Die über die Netzwerkschnittstelle ausgetauschten Nachrichten enthalten daher einen Ethernet-Paketkopf (18 Byte), einen IP-Paketkopf (20 Byte), einen UDP-Paketkopf (8 Byte) und den Paketkopf des unzuverlässigen Protokolls (11 Byte). Der bei der Übertragung der Nutzdaten verursachte Overhead beträgt also 77 Byte (inkl. des 10 Gigabit Ethernet-spezifischen *Interframe Gap* und der Präambel). Im Falle von 0 Byte Nutzdaten werden die Nachrichten bei dem Versand durch die Netzwerkschnittstelle auf die Minimallänge von 84 Byte verlängert.

Im Folgenden werden die Ergebnisse der Leistungsmessungen vorgestellt. Zunächst wird der *Basisfall* und anschließend der Anwendungsfall *UDP/Custom* betrachtet.

6.8.5.1 Basisfall unidirektional

Im ersten Experiment wurde im Basisfall die Sendeleistung und anschließend die Empfangsleistung untersucht. Die Ergebnisse werden in Abbildung 6.13 gezeigt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der y-Achse wird die Paketrate in Millionen Pakete pro Sekunde (MPPS) angegeben. Die Kurve *Maximum* zeigt die maximal mögliche Paketrate der Netzwerkschnittstelle in Abhängigkeit von der Nutzdatengröße. *DPDK RX* und *DPDK TX* stellen die gemessene Empfangs- und Sendeleistung dar. Zum Vergleich zeigen *Socket RX* und *Socket TX* die Empfangs- und Sendeleistung, die mit der auf der Socket-API basierenden Version von NENA gemessen wurde. Die Fehlerbalken geben für jede Nutzdatengröße die minimale, maximale und durchschnittliche (arithmetisches Mittel) Sende- bzw. Empfangsleistung in den zehn Versuchsläufen an.

Wie in der Abbildung zu sehen ist, erreicht DPDK-NENA beim Senden bei jeder untersuchten Nutzdatengröße die maximal mögliche Paketrate der Netzwerkschnittstelle.

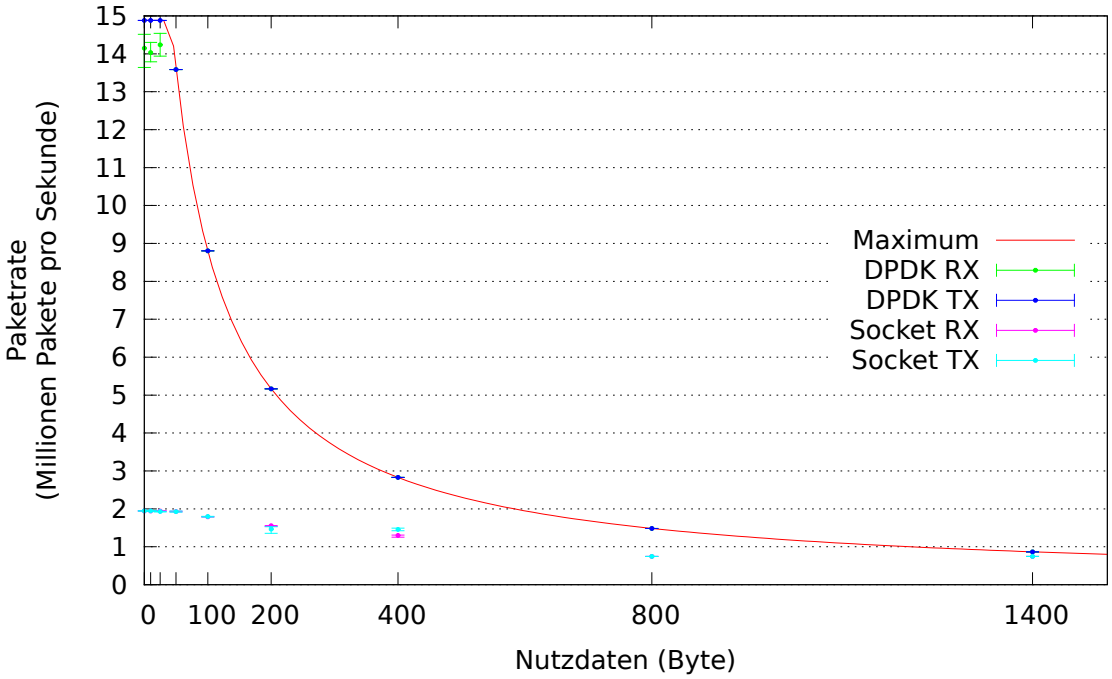

Abbildung 6.13: *Basisfall: Paketraten bei der unidirektionalen Übertragung.*

Beim Empfangen erreicht DPDK-NENA eine maximale Paketrate von etwa 14.5 MPPS, was knapp unter der maximal möglichen Paketrate von 14.88 MPPS liegt. Die Kommunikation zwischen DPDK-NENA und der Anwendung über den gemeinsamen Ring-Puffer stellt dabei den Flaschenhals dar. Verwirft man die empfangenen Nachrichten in DPDK-NENA, anstatt sie an die Anwendung weiterzureichen, erreicht man auch beim Empfang die maximale Paketrate. Ab einer Nutzdatengröße von 50 Byte erreicht DPDK-NENA die Empfangsleistung der Netzwerkschnittstelle.

Die Socket-basierte Version von NENA erreicht sowohl beim Senden als auch beim Empfangen eine maximale Paketrate von ca. 1.95 MPPS, was deutlich unter den möglichen 14.88 MPPS liegt. Zudem wird mit keiner Nutzdatengröße die maximal mögliche Paketrate erreicht. DPDK-NENA erzielt demnach in diesem Experiment eine etwa 7.6-mal höhere Paketrate als die Socket-basierte Version von NENA.

Die in diesem Experiment erzielten Datenraten werden in Abbildung 6.14 dargestellt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der y-Achse wird die Datenrate in Gigabit/s angegeben. Die maximal mögliche Datenrate wird durch eine rote Linie angedeutet. Die Kurven *DPDK RX* und *DPDK TX* stellen die ermittelte Datenrate beim Empfänger und beim Sender dar. Zum Vergleich werden die mit der Socket-basierten Version von NENA erreichten Datenraten mit den Kurven *Socket RX* und *Socket TX* gezeigt. Die Fehlerbalken geben jeweils die minimale, maximale und durchschnittliche (arithmetisches Mittel) Datenrate in den zehn Versuchsläufen an.

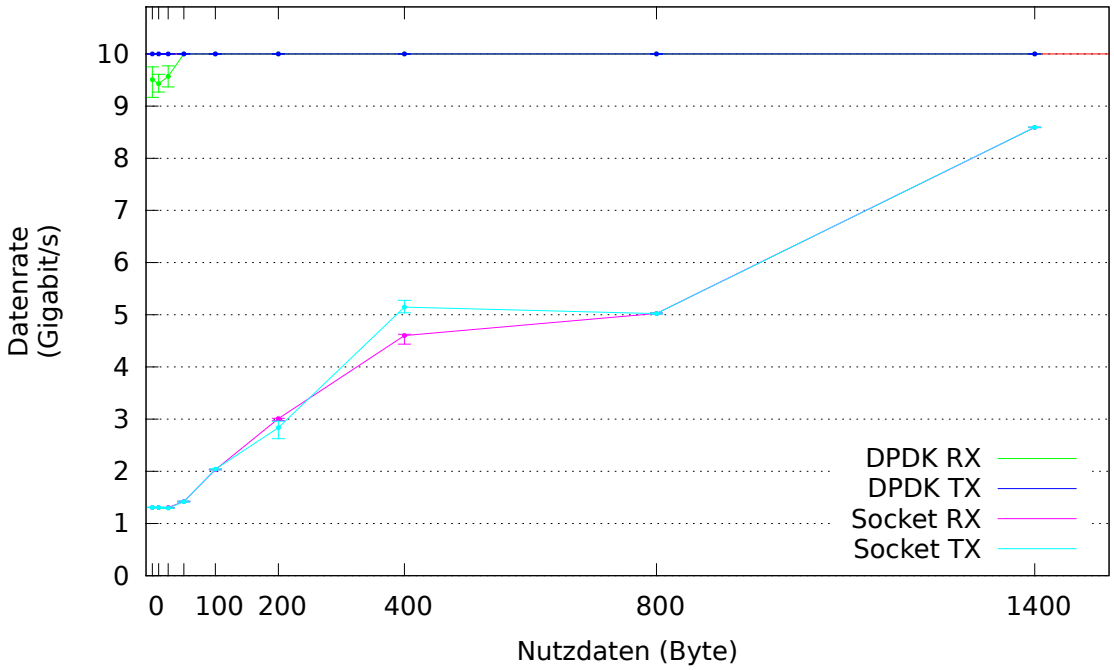

Abbildung 6.14: *Basisfall: Datenraten bei der unidirektionalen Übertragung.*

DPDK-NENA erreicht beim Senden eine Datenrate von 10 Gigabit/s mit allen untersuchten Nutzdatengrößen und lastet damit die Netzwerkschnittstelle vollständig aus. Beim Empfangen erreicht DPDK-NENA bereits mit der kleinsten Nutzdatengröße eine Datenrate von etwa 9.5 Gigabit/s und lastet damit die Netzwerkschnittstelle mit ca. 95 % aus. Ab 50 Byte erreicht DPDK-NENA beim Empfang die volle Datenrate der Netzwerkschnittstelle.

Die erzielte Datenrate verdeutlicht, dass die Socket-basierte Version von NENA nicht die maximale Leistung der Netzwerkschnittstelle erreicht. Es wird eine maximale Datenrate von 8.6 Gigabit/s und damit eine maximale Auslastung von 86 % erreicht. Mit den kleinsten Nutzdatengrößen wird sogar nur eine Auslastung von etwa 13 % erreicht. Da die Paketrate bei der Kommunikation über das Betriebssystem in Abhängigkeit von der Nutzdatengröße abnimmt, wächst außerdem die Datenrate vergleichsweise langsam. Bei der Nutzdatengröße 800 Byte tritt sogar der Fall ein, dass die Datenrate im Vergleich zu der Nutzdatengröße von 400 Byte nicht ansteigt, sondern im Mittel leicht abfällt.

6.8.5.2 Basisfall bidirektional

Im zweiten Experiment wurde beim Basisfall die Sendeleistung und die Empfangsleistung bei einer gleichzeitigen Übertragung in beide Richtungen untersucht. Die Ergebnisse werden in Abbildung 6.15 gezeigt. Auf der x-Achse wird die Größe der Nutzdaten in

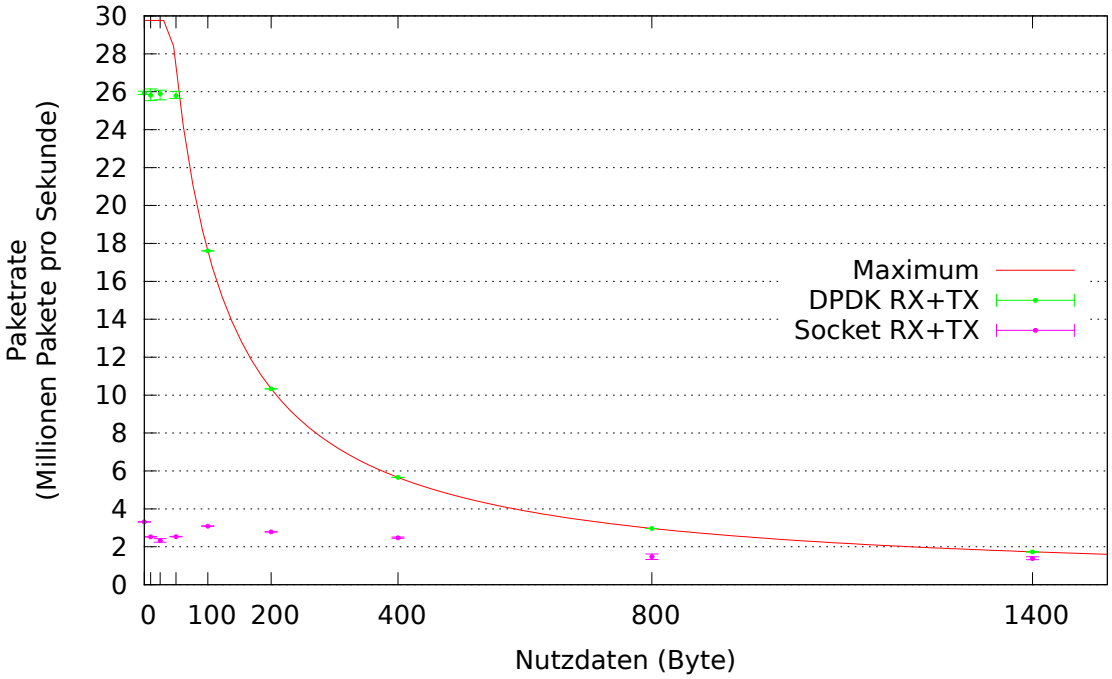

Abbildung 6.15: *Basisfall: Paketraten bei der bidirektionalen Übertragung.*

Byte angegeben. Auf der y-Achse wird die Paketrate in Millionen Pakete pro Sekunde (MPPS) angegeben. Die Kurve *Maximum* zeigt die maximal mögliche Paketrate der Netzwerkschnittstelle in Abhängigkeit von der Nutzdatengröße. *DPDK RX+TX* stellt die kombinierte Empfangs- und Sendeleistung mit DPDK-NENA dar. Zum Vergleich zeigt *Socket RX+TX* die mit der Socket-basierten Version von NENA gemessene Empfangs- und Sendeleistung. Die Fehlerbalken geben für jede Nutzdatengröße die minimale, maximale und durchschnittliche (arithmetisches Mittel) Sende- bzw. Empfangsleistung in den zehn Versuchsläufen an.

Die maximal mögliche Paketrate über die Netzwerkschnittstelle beträgt 29.76 MPPS. DPDK erreicht eine maximale Paketrate von ca. 26 MPPS. Untersuchungen in [26] weisen auf eine Beschränkung in der Hardware der Netzwerkschnittstelle hin, die auch für die Evaluation von DPDK-NENA verwendet wurde. Daher wird davon ausgegangen, dass die Hardware den Flaschenhals in diesem Experiment darstellt. Ab einer Nutzdatengröße von 50 Byte wird die maximal mögliche Paketrate fast erreicht. Die weiteren Werte entsprechen dem Maximum der Netzwerkschnittstelle.

Die Socket-basierte Version von NENA erreicht eine maximale Paketrate von ca. 3.1 MPPS, was deutlich unter den möglichen 29.76 MPPS liegt. Zudem wird mit keiner Nutzdatengröße die maximal mögliche Paketrate erreicht. DPDK-NENA erzielt also in diesem Experiment eine etwa 8.4-mal höhere Paketrate als die Socket-basierte Version von NENA. Darüber hinaus weisst die Socket-basierte Version von NENA durch den

Datenaustausch über das Betriebssystem Unregelmäßigkeiten bei der Übertragung von Nachrichten auf.

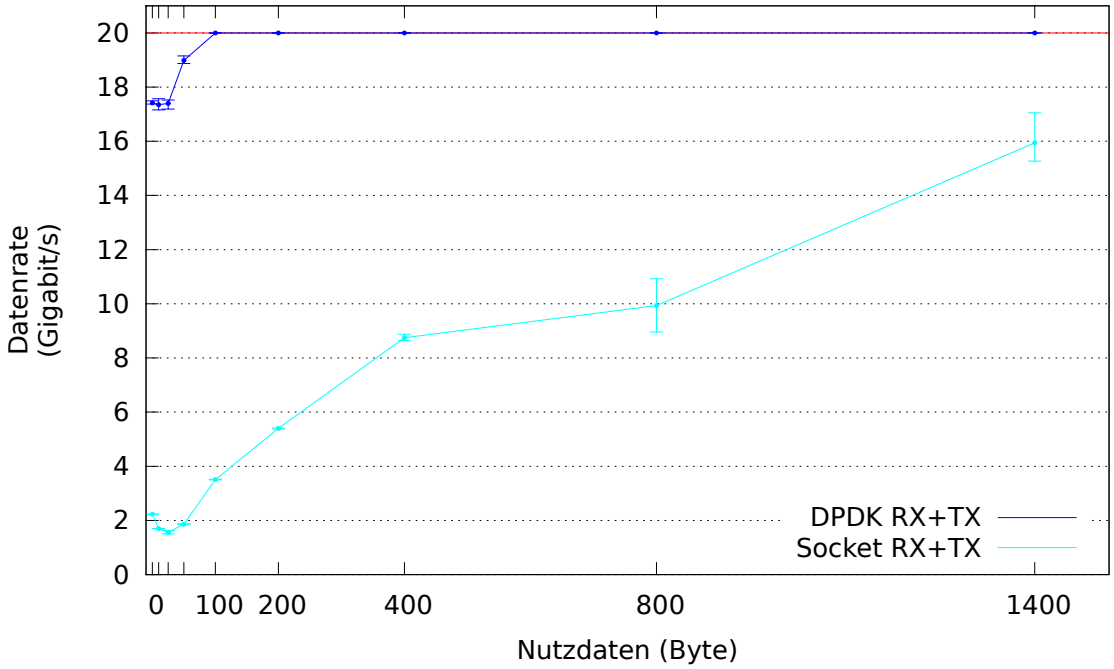

Abbildung 6.16: *Basisfall: Datenraten bei der bidirektionalen Übertragung.*

Die in diesem Experiment erzielten Datenraten werden in Abbildung 6.16 dargestellt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der y-Achse wird die Datenrate in Gigabit/s angegeben. Die maximal mögliche Datenrate wird durch eine rote Linie angedeutet. Die Kurve *DPDK RX+TX* stellt die kombinierte Datenrate beim Senden und Empfangen mit DPDK-NENA dar. Zum Vergleich wird die mit der Socket-API erreichte Datenrate mit der Kurve *Socket RX+TX* gezeigt. Die Fehlerbalken geben für jede Nutzdatengröße die minimale, maximale und durchschnittliche (arithmetisches Mittel) Datenrate in den zehn Versuchsläufen an.

DPDK-NENA erreicht mit der kleinsten Nutzdatengröße bereits eine Datenrate von etwa 17.4 Gigabit/s. Bei einer Nutzdatengröße von 50 Byte wird eine Datenrate von ca. 19 Gigabit/s erzielt. Bei den weiteren Nutzdatengrößen erreicht DPDK-NENA die maximale Datenrate der Netzwerkschnittstelle und lastet damit die Netzwerkschnittstelle vollständig aus.

Die erzielte Datenrate verdeutlicht, dass die Socket-basierte Version von NENA nicht die maximale Leistung der Netzwerkschnittstelle erreicht. Es wird eine maximale Datenrate von 17 Gigabit/s erreicht. Mit den kleinsten Nutzdatengrößen wird sogar nur eine Datenrate von 2.2 Gigabit/s erreicht. Zudem verdeutlicht die Datenrate die durch den

Datenaustausch über das Betriebssystem auftretenden Unregelmäßigkeiten. Die Paketraten, und damit auch die Datenraten, sinken zunächst mit zunehmender Nutzdatengröße, bevor sie ab etwa 50 Byte Nutzdaten wieder steigen. Ab 100 Byte Nutzdaten sinkt die Paketrate wieder mit zunehmender Nutzdatengröße und die Datenrate wächst schwächer. Außerdem beginnen ab 800 Byte Nutzdaten die Messwerte stärker zu schwanken.

6.8.5.3 UDP/Custom unidirektional

Im dritten Experiment wurde die Sendeleistung und anschließend die Empfangsleistung im Anwendungsfall *UDP/Custom* untersucht. Die Ergebnisse werden in Abbildung 6.17 gezeigt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der

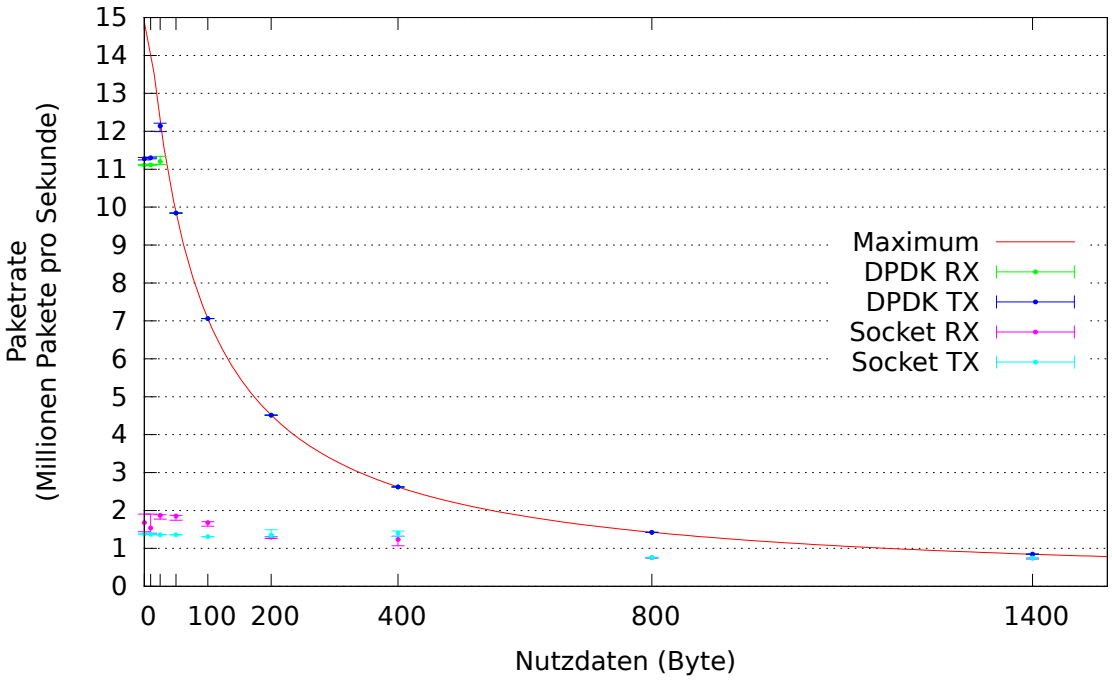

Abbildung 6.17: *UDP/Custom: Paketraten bei der unidirektionalen Übertragung.*

y-Achse wird die Paketrate in Millionen Pakete pro Sekunde (MPPS) angegeben. Die Kurve *Maximum* zeigt die maximal mögliche Paketrate der Netzwerkschnittstelle in Abhängigkeit von der Nutzdatengröße. *DPDK RX* und *DPDK TX* stellen die gemessene Empfangs- und Sendeleistung dar. Zum Vergleich werden die mit der Socket-basierten Version von NENA gemessene Empfangs- und Sendeleistung mit *Socket RX* und *Socket TX* gezeigt. Die Fehlerbalken geben für jede Nutzdatengröße die minimale, maximale und durchschnittliche (arithmetisches Mittel) Sende- bzw. Empfangsleistung in den zehn Versuchsläufen an.

DPDK-NENA erreicht anfangs beim Senden und beim Empfangen eine maximale Paketrate von etwa 11.3 MPPS. Mit der Nutzdatengröße 25 Byte erhöht sich die Paketrate beim Senden auf etwa 12.1 MPPS und erreicht damit in etwa die maximal mögliche Paketrate der Netzwerkschnittstelle. Beim Empfangen wird dennoch keine höhere Paketrate als 11.3 MPPS erreicht. Der Nachrichtenaustausch zwischen dem Modul für Virtuelle Verbindungen und der Basiskomponente über den gemeinsamen Ring-Puffer stellt in diesem Fall den Flaschenhals beim Senden dar. Verringert man durch die Parametrisierung dieses Ring-Puffers den Synchronisationsaufwand (siehe Abschnitt 6.7) erhöht sich die Paketrate beim Senden auf ca. 14 MPPS. Mit den weiteren Nutzdatengrößen erreicht DPDK-NENA die maximal mögliche Paketrate der Netzwerkschnittstelle sowohl beim Senden als auch beim Empfangen.

Die Socket-basierte Version von NENA erreicht beim Senden etwa bis zu 1.5 MPPS und beim Empfangen bis zu 1.9 MPPS. Außerdem wird auch in diesem Experiment bei keiner Nutzdatengröße die maximale Paketrate erreicht. Auffällig ist, dass bei den geringen Nutzdatengrößen beim Empfang höhere Paketraten gemessen wurden als beim Senden. Dabei muss allerdings beachtet werden, dass die Sende- und Empfangstests in diesem Experiment zu unterschiedlichen Zeitpunkten durchgeführt wurden. Dies zeigt wiederum wie unterschiedlich die Leistung beim Datenaustausch über das Betriebssystem ausfallen kann.

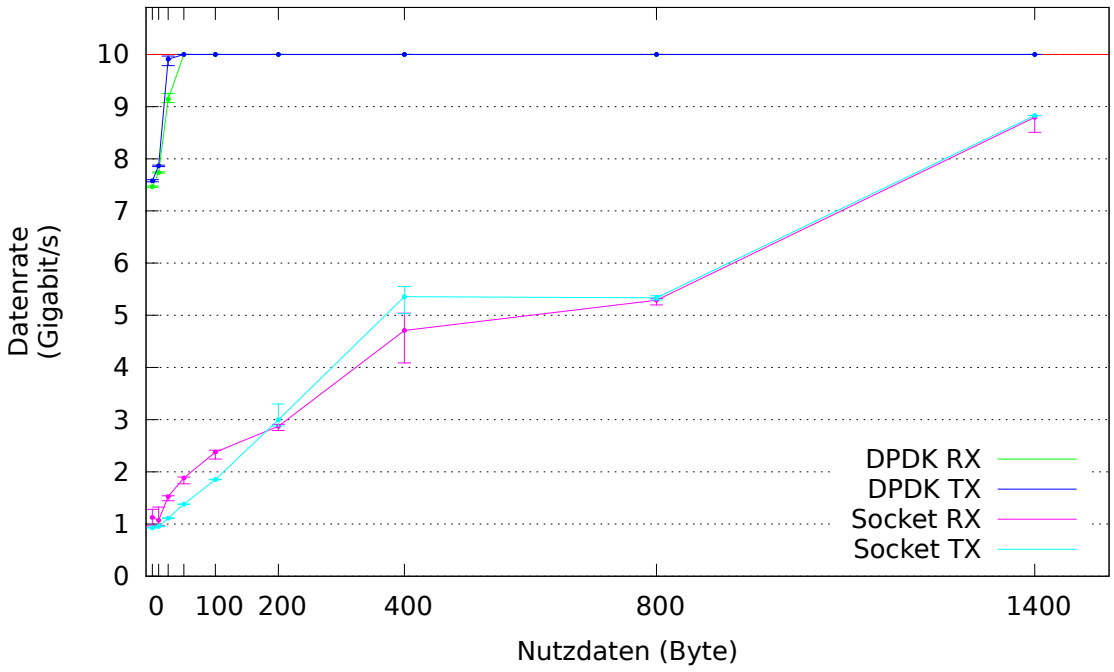

Abbildung 6.18: *UDP/Custom: Datenraten bei der unidirektionalen Übertragung.*

Die in diesem Experiment erzielten Datenraten werden in Abbildung 6.18 dargestellt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der y-Achse wird die Datenrate in Gigabit/s angegeben. Die maximal mögliche Datenrate wird durch eine rote Linie angedeutet. Die Kurven *DPDK RX* und *DPDK TX* stellen die ermittelte Datenrate beim Empfänger und beim Sender dar. Zum Vergleich werden die mit der Socket-API erreichten Datenraten mit den Kurven *Socket RX* und *Socket TX* gezeigt. Die Fehlerbalken geben für jede Nutzdatengröße die minimale, maximale und durchschnittliche (arithmetisches Mittel) Datenrate in den zehn Versuchsläufen an.

DPDK-NENA erreicht beim Senden und Empfangen mit der minimalen Nutzdatengröße eine Datenrate von etwa 7.6 Gigabit/s. Bei einer Nutzdatengröße von 25 Byte wird beim Senden ca. 9.9 Gigabit/s und beim Empfangen 9.1 Gigabit/s erreicht. Mit den weiteren Nutzdatengrößen wird eine Datenrate von 10 Gigabit/s erreicht und die Netzwerkschnittstelle vollständig ausgelastet.

Die Socket-basierte Version von NENA erreicht mit der minimalen Nutzdatengröße nur eine Datenrate von 1,3 Gigabit/s. Es wird nur eine maximale Datenrate von 8.7 Gigabit/s und damit eine Auslastung von 87 % erzielt.

6.8.5.4 UDP/Custom bidirektional

Im letzten Experiment wurde die Sendeleistung und die Empfangsleistung bei einer gleichzeitigen Übertragung in beide Richtungen im Anwendungsfall *UDP/Custom* untersucht. Die Ergebnisse werden in Abbildung 6.19 gezeigt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der y-Achse wird die Paketrate in Millionen Pakete pro Sekunde (MPPS) angegeben. Die Kurve *Maximum* zeigt die maximal mögliche Paketrate der Netzwerkschnittstelle. *DPDK RX+TX* stellt die kombinierte Empfangs- und Sendeleistung dar. Zum Vergleich werden die mit der Socket-API gemessene Empfangs- und Sendeleistung mit *Socket RX+TX* gezeigt. Die Fehlerbalken geben die minimale, maximale und durchschnittliche (arithmetisches Mittel) Sende- bzw. Empfangsleistung in zehn Versuchsläufen an.

Die maximal mögliche Paketrate über die Netzwerkschnittstelle bei der bidirektionalen Datenübertragung beträgt 29.76 MPPS. DPDK-NENA erreicht eine maximale Paketrate von ca. 25.4 MPPS. Ab einer Nutzdatengröße von 25 Byte wird die maximal mögliche Paketrate der Netzwerkschnittstelle erreicht.

Die Socket-basierte Version von NENA erreicht eine maximale Paketrate von ca. 3.1 MPPS, was deutlich unter den möglichen 29.76 MPPS liegt. Zudem wird mit einer Nutzdatengröße von 1400 Byte in einzelnen Versuchsläufen die maximal mögliche Paketrate erreicht. DPDK-NENA erzielt demnach in diesem Experiment eine etwa 8.2-mal höhere Paketrate als die Socket-basierte Version von NENA.

Die in diesem Experiment erzielten Datenraten werden in Abbildung 6.20 dargestellt. Auf der x-Achse wird die Größe der Nutzdaten in Byte angegeben. Auf der y-Achse wird

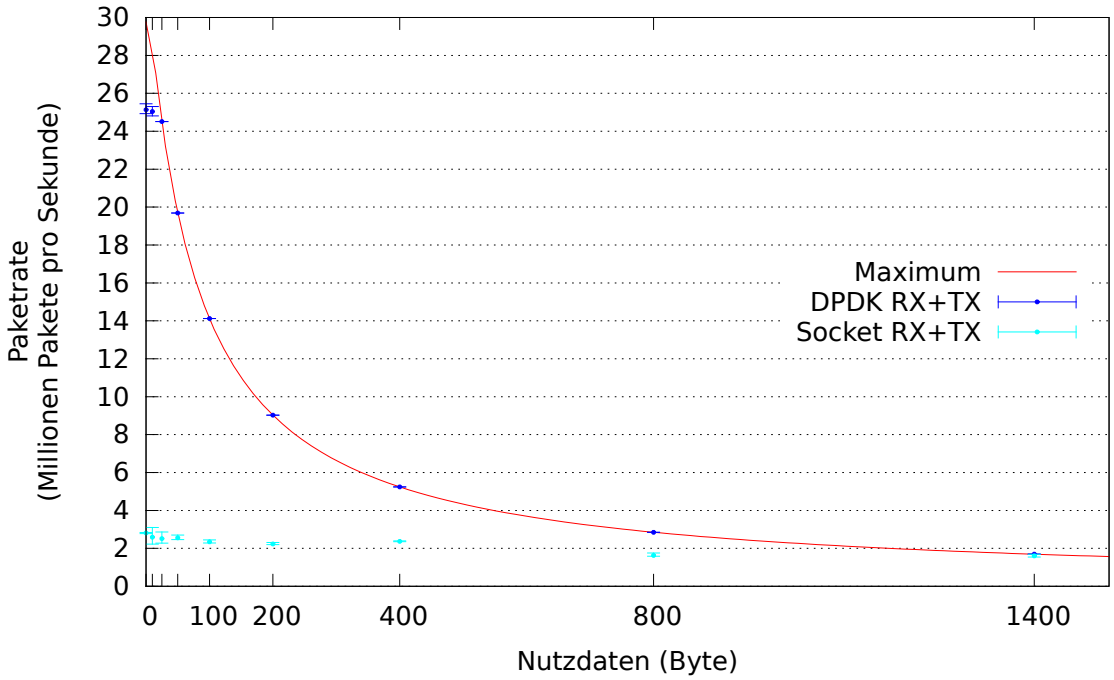

Abbildung 6.19: *UDP/Custom: Paketraten bei der bidirektionalen Übertragung.*

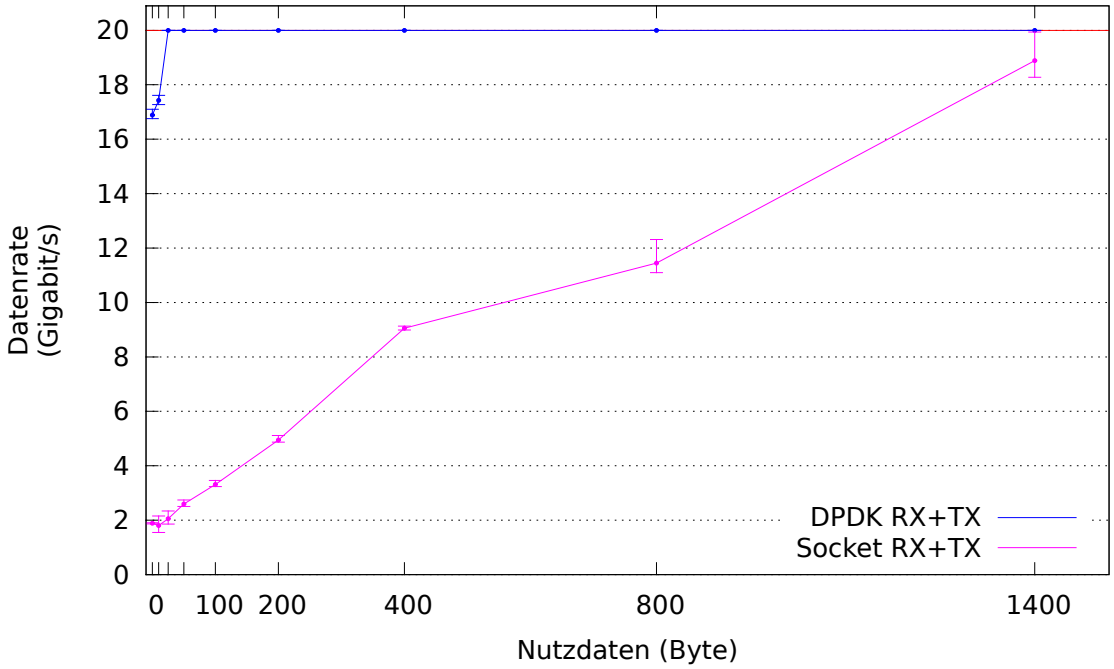

Abbildung 6.20: *UDP/Custom: Datenrate bei der bidirektionalen Übertragung.*

die Datenrate in Gigabit/s angegeben. Die maximal möglichen Datenraten werden durch eine rote Linie angedeutet. Die Kurve *DPDK RX+TX* stellt die mit DPDK-NENA erreichten Datenraten dar. Zum Vergleich werden die mit der Socket-API erreichten Datenraten mit der Kurve *Socket RX+TX* gezeigt.

DPDK-NENA erreicht mit der kleinsten Nutzdatengröße bereits eine Datenrate von etwa 17 Gigabit/s. Ab einer Nutzdatengröße von 25 Byte wird die maximale Datenrate der Netzwerkschnittstelle erzielt und damit die Netzwerkschnittstelle vollständig ausgelastet.

Die Socket-basierte Version von NENA erreicht mit den kleinsten Nutzdatengrößen nur eine Datenrate von 2.2 Gigabit/s. Bei einer Nutzdatengröße von 1400 Byte wird in einzelnen Versuchsläufen die maximale Datenrate der Netzwerkschnittstelle erreicht.

6.8.6 Fazit

Die in diesem Abschnitt gezeigten Leistungsmessungen belegen, dass DPDK-NENA in der Lage ist, eine mehr als acht mal höhere Paketrate zu erreichen als ein auf der Socket-API basierendes NENA. Außerdem kann DPDK-NENA eine 10 Gigabit/s Netzwerkschnittstelle bereits bei kleinen Paketlängen auslasten, was mit der Socket-API basierten Version in den Versuchen nicht möglich war. Dieser Leistungsunterschied zwischen den evaluierten NENA-Varianten wird durch die Optimierung des Datenaustauschs zwischen NENA und dem Netz erreicht. Somit konnte gezeigt werden, wie wichtig ein schneller Datenaustausch über Netzwerkschnittstellen für eine leistungsfähige Nutzung von DsNs auf Endsystemen ist.

6.9 Zusammenfassung

In diesem Kapitel wurde DPDK-NENA, ein Rahmenwerk für einen leistungsfähigen Datenaustausch zwischen Endsystemen und DsNs, vorgestellt. DPDK-NENA verwendet geteilten Speicher, um mit Anwendungen Daten auszutauschen. Außerdem greift es direkt auf die Netzwerkschnittstellen des Endsystems zu. Die Nachrichtenverarbeitung in DPDK-NENA wird auf mehrere Prozessor-Kerne aufgeteilt und durch den geteilten Speicher können Nachrichten zwischen Anwendungen und Netzwerkschnittstellen ohne zusätzliches Kopieren ausgetauscht werden. Sowohl die Protokolle der DsNs als auch die Protokolle für die Virtuellen Verbindungen zu DsNs können zur Laufzeit ausgetauscht werden. In der Evaluierung wurde gezeigt, dass DPDK-NENA eine 10 Gigabit/s Netzwerkschnittstelle schon mit kleinen Paketen auslasten kann und mehr als acht mal höhere Paketraten als eine auf der Socket-API basierende Version erreicht. DPDK-NENA kombiniert somit die Flexibilität von DsNs mit hoher Leistung und stellt ein leistungsfähiges Rahmenwerk für die Nutzung von DsNs auf Endsystemen dar.

Kapitel 7

Zusammenfassung und Ausblick

Heutzutage stellen Dienste wie z.B. Telefonie, Video-Streaming, Online-Banking oder Datei-Austausch im Internet unterschiedlichste Anforderungen an Kommunikationsprotokolle. Es gibt kein Protokoll, das Anforderungen wie niedrige Latenzen, hohe Datenraten, Energie-Sparen oder die Bewahrung der Privatsphäre ideal unterstützt. Eine Vielzahl spezialisierter Protokolle ist notwendig. Allerdings ist die Ausbringung neuer Protokolle im Internet schwierig und zeitaufwendig, was z.B. durch die bisher geringe Verfügbarkeit von Protokollen wie IPv6, SCTP und DCCP verdeutlicht wird. Dies hat zur Entwicklung von Dienst-spezifischen Netzen geführt.

Dienst-spezifische Netze (DsN) sind virtuelle Netze, die über die Netz-Infrastruktur betrieben werden. In einem DsN wird ein Dienst realisiert und das Dienst-spezifische Netz ist an die Anforderungen des Dienstes angepasst. Innerhalb des DsN werden hierfür Dienst-spezifische Protokolle verwendet, die an die Eigenschaften und Anforderungen des erbrachten Dienstes angepasst sind. Dienst-spezifische Netze bieten einen hohen Grad an Flexibilität. Dank Virtualisierungstechnologien ist eine schnelle Ausbringung Dienst-spezifischer Netze möglich und eine Interoperabilität zwischen Dienst-spezifischen Protokollen unterschiedlicher DsNs ist nicht nötig. Außerdem ist eine Weiterentwicklung der Netz-Infrastruktur unabhängig von Dienst-spezifischen Netzen möglich, so lange Konnektivität zu Dienst-spezifischen Netzen erhalten bleibt. Neue Protokolle können in der Netz-Infrastruktur ausgebracht werden, ohne Dienste oder Dienst-spezifische Protokolle innerhalb der Dienst-spezifischen Netze anpassen zu müssen.

Allerdings existierten noch Probleme, die gelöst werden mussten, um Dienst-spezifische Netze tatsächlich nutzen zu können. Eine offene Fragestellung war, wie Endsysteme auf Dienst-spezifische Netze zugreifen und diese effizient nutzen können. Besondere Herausforderungen stellten die Heterogenität und Dynamik Dienst-spezifischer Netze dar. Ein Endsystem muss für die Nutzung von Diensten Virtuelle Verbindungen zu Dienst-spezifischen Netzen aufbauen und betreiben. In den Dienst-spezifischen Netzen

werden verschiedene, an die Dienste angepasste Protokolle verwendet, über die das Endsystem kein a priori Wissen besitzt. Das Endsystem muss diese laden und verwenden können. Der Betrieb Dienst-spezifischer Netze und Dienst-spezifischer Protokolle benötigt zudem Ressourcen auf einem Endsystem, die reguliert werden müssen, um eine negative gegenseitige Beeinflussung von Dienst-spezifischen Netzen auf einem Endsystem zu vermeiden. Nutzen Anwendungen Dienste mit hohen Anforderungen an den Durchsatz, müssen sie dabei auch einen hohen Durchsatz erreichen, unabhängig davon, welche Paketlängen von den entsprechenden Dienst-spezifischen Protokollen verwendet werden. Vor allem das Erreichen eines hohen Durchsatzes bei geringen Paketlängen stellt eine besondere Herausforderung dar.

Das existierende Rahmenwerk NENA unterstützt die Verwendung Virtueller Verbindungen sowie das Laden und Ausführen Dienst-spezifischer Protokolle auf einem Endsystem. Allerdings mussten diese Virtuellen Verbindungen bisher manuell konfiguriert und aufgebaut werden. Ebenso mussten die Dienst-spezifischen Protokolle für die Dienst-spezifischen Netze, auf die das Endsystem zugreift, manuell bezogen und geladen werden. Außerdem wurde nicht betrachtet, wie die für Dienst-spezifische Netze verwendeten Ressourcen auf einem Endsystem reguliert werden können. Darüber hinaus wurde bei der Entwicklung des ursprünglichen NENA kein Fokus auf hohe Leistung gelegt, weswegen es keinen schnellen Datenaustausch zwischen Anwendungen und Dienst-spezifischen Netzen ermöglicht.

Aus diesen Gründen wurden im Rahmen dieser Arbeit Lösungen für eine flexible und leistungsfähige Nutzung von Dienst-spezifischen Netzen auf Endsystemen entwickelt. Die Beiträge dieser Arbeit lassen sich folgendermaßen gliedern:

- *Zugriffsverfahren auf Dienst-spezifische Netze für Endsysteme*

  Das Zugriffsverfahren für Dienst-spezifische Netze erlaubt einen flexiblen Zugriff auf Dienst-spezifische Netze. Es ermöglicht Endsystemen, Dienst-spezifische Netze für Dienste zu finden, die benötigten Protokolle zu beziehen und Verbindungen zu Dienst-spezifischen Netzen aufzubauen, wenn der Nutzer auf Dienste zugreift. Das Zugriffsverfahren löst diese Probleme transparent für den Nutzer und erlaubt, beliebige Methoden für das Aufbauen Virtueller Verbindungen und das Beziehen von Dienst-spezifischen Protokollen zu verwenden.

- *Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen auf Endsystemen*

  Die Verwaltung der für Dienst-spezifische Netze verwendete Ressourcen wird durch das, im Rahmen dieser Arbeit entworfene, Hierarchische Knoten-Management erreicht. Es ermöglicht eine flexible Verwaltung der für Dienst-spezifische Netze benötigten Netz-, Speicher-, und Prozessor-Ressourcen auf einem Endsystem. Die Ressourcen-Verwaltung erlaubt, die auf einem Endsystem verfügbaren Ressourcen

auf verschiedene Dienst-spezifische Netze und die darin verwendeten Protokolle aufzuteilen und dabei Nutzer-Prioritäten zu beachten.

- *Leistungsfähiges Rahmenwerk für Dienst-spezifische Netze auf Endsystemen*

  Das in dieser Arbeit entwickelte leistungsfähige Rahmenwerk für Dienst-spezifische Netze kombiniert die Flexibilität Dienst-spezifischer Netze mit einer hohen Leistung. Durch die Verwendung eines modularen Aufbaus ermöglicht das Rahmenwerk den Betrieb Dienst-spezifischer Protokolle und Virtueller Verbindungen. Dabei erreicht es durch Optimierungen, wie den Datenaustausch ohne Kopieren, die Verwendung mehrerer Prozessor-Kerne und einen direkten Zugriff auf Netzwerkschnittstellen, hohe Paketraten selbst bei kleinen Paketen.

In den folgenden beiden Abschnitten werden die Beiträge dieser Arbeit zusammengefasst und ein Ausblick auf weiterführende Arbeiten gegeben.

7.1 Beiträge dieser Arbeit

(1) *Zugriffsverfahren auf Dienst-spezifische Netze für Endsysteme*: Das in dieser Arbeit entwickelte Zugriffsverfahren erlaubt Endsystemen, Dienst-spezifische Netze für Dienste zu finden, Dienst-spezifische Netze auszuwählen, die Dienst-spezifischen Protokolle Dienst-spezifischer Netze zu beziehen und Virtuelle Verbindungen zu Dienst-spezifischen Netzen aufzubauen. Das Zugriffsverfahren verwendet Komponenten wie einen Mapping-Dienst und einen Speicher-Dienst. Ein Dienste-Anbieter legt für sein Dienst-spezifisches Netz die Verbindungsmethoden und die verwendeten Dienst-spezifischen Protokolle fest. Im Speicher-Dienst legt er die Dienst-spezifischen Protokolle ab. Im Mapping-Dienst legt er die für den Zugriff auf das DsN benötigten Informationen ab. Die Endsysteme rufen beim Zugriff auf den Dienst diese Informationen aus dem Mapping-Dienst ab und nutzen sie, um die Dienst-spezifischen Protokolle zu beziehen und eine Virtuelle Verbindung zum Dienst-spezifischen Netz aufzubauen. Das Zugriffsverfahren erreicht Flexibilität dadurch, dass sowohl die Methoden für das Beziehen der Dienst-spezifischen Protokolle als auch die Methoden für das Aufbauen Virtueller Verbindungen ausgetauscht werden können. Außerdem erlaubt es den Dienste-Anbietern, für ihre Dienst-spezifischen Netze die Bezugs- und Verbindungsmethoden selbst festzulegen. Endsysteme können aus den Vorgaben des Dienste-Anbieters basierend auf den verfügbaren Methoden geeignete Bezugs- und Verbindungsmethoden auswählen. Die Funktionsfähigkeit des Zugriffsverfahren wurde mit einem Demonstrator gezeigt und der, durch das Zugriffsverfahren verursachte, Zeitaufwand beim Zugriff auf Dienste wurde evaluiert. Dabei wurde gezeigt, dass der zeitliche Aufwand beim Zugriff auf ein beispielhaftes Dienst-spezifisches Netz nur wenige Sekunden beträgt. Das Zugriffsverfahren ermöglicht Endsystemen daher erstmals einen flexiblen und nutzerfreundlichen Zugriff auf Dienst-spezifische Netze.

(2) *Verwaltung der für Dienst-spezifische Netze verwendeten Ressourcen auf Endsystemen*: Das in dieser Arbeit entworfene Hierarchische Knoten-Management erlaubt, die auf dem Endsystem verwendeten Dienst-spezifischen Netze bzw. Dienst-spezifischen Protokolle zur Laufzeit an die Netz-Eigenschaften und die verfügbaren Ressourcen anzupassen. Die auf dem Endsystem verfügbaren Prozessor-, Speicher- und Netz-Ressourcen können den Dienst-spezifischen Netzen und Dienst-spezifischen Protokollen zugewiesen und entzogen werden. Dem Nutzer wird ermöglicht, das Hierarchische Knoten-Management und die Ressourcen-Verwaltung durch Richtlinien zu beeinflussen. Das Hierarchische Knoten-Management ist in verschiedene Verwaltungskomponenten gegliedert und in einer Baum-förmigen Hierarchie organisiert. Eine Knoten-weite Verwaltungskomponente bildet die Wurzel des Baumes. Auf der Ebene darunter befinden sich für jedes DsN eine Netz-spezifische Verwaltungskomponente und auf der untersten Ebene für jedes Protokoll eine Protokoll-spezifische Verwaltungskomponente. Entlang der Hierarchie tauschen die Verwaltungskomponenten Informationen, Ressourcen und Richtlinien aus. Durch den modularen Aufbau und die Hierarchie wird Flexibilität erreicht. Ändert sich die Menge der Dienst-spezifischen Netze, auf die das Endsystem zugreift, können die entsprechenden Verwaltungskomponenten einfach entfernt oder hinzugefügt werden, ohne das gesamte Verwaltungssystem anpassen zu müssen. Da eigene Verwaltungskomponenten für die Dienst-spezifischen Netze und Protokolle entworfen werden, erlauben sie eine an deren Anforderungen und Eigenschaften angepasste Verwaltung. Das Hierarchische Knoten-Management wurde mit einem Prototypen evaluiert. Dabei wurde gezeigt, dass es in der Lage ist, sowohl die Komponenten Dienst-spezifischer Netze an die Eigenschaften der Netz-Infrastruktur anzupassen als auch die beschränkten Ressourcen des Endsystems zu verwalten. Das Hierarchische Knoten-Management stellt also ein flexibles Verwaltungssystem für Dienst-spezifische Netze auf Endsystemen dar und erreicht eine flexible Verwaltung der Ressourcen von Endsystemen.

(3) *Leistungsfähiges Rahmenwerk für Dienst-spezifische Netze auf Endsystemen*: In dieser Arbeit wurde DPDK-NENA, ein Rahmenwerk für einen leistungsfähigen Datenaustausch zwischen Endsystemen und Dienst-spezifischen Netzen, entworfen. DPDK-NENA wurde von Grund auf neu entwickelt und verwendet geteilten Speicher, um mit Anwendungen Daten auszutauschen. Außerdem greift es direkt auf die Netzwerkschnittstellen des Endsystems zu. Die Nachrichtenverarbeitung in DPDK-NENA wird auf mehrere Prozessor-Kerne aufgeteilt und durch den geteilten Speicher können Nachrichten zwischen Anwendungen und Netzwerkschnittstellen ohne zusätzliches Kopieren ausgetauscht werden. Sowohl die Protokolle der Dienst-spezifischen Netze als auch die Protokolle für die Virtuellen Verbindungen zu DsNs können zur Laufzeit ausgetauscht werden. DPDK-NENA wurde mit Hilfe eines Prototypen evaluiert. Dabei wurde gezeigt, dass DPDK-NENA eine 10 Gigabit/s Netzwerkschnittstelle bereits mit kleinen Paketen auslasten kann und mehr als acht mal höhere Paketraten als eine auf der Socket-API basierende Version von NENA erreicht. DPDK-NENA kombiniert somit ein hohes Maß

an Flexibilität mit hoher Leistung und stellt ein leistungsfähiges Rahmenwerk für die Nutzung Dienst-spezifischer Netze auf Endsystemen dar.

7.2 Ausblick auf weiterführende Arbeiten

Die in dieser Arbeit vorgestellten Lösungen erlauben eine flexible und leistungsfähige Nutzung Dienst-spezifischer Netze auf Endsystemen. Aufbauend auf diesen Lösungen können weitere Herausforderungen Dienst-spezifischer Netze auf der Seite des Dienste-Anbieters angegangen werden. Eine Herausforderung ist die Ausbringung der Dienst-spezifischen Netze und die Anpassung eines DsN zur Laufzeit an die aktuelle, durch den Zugriff von Endsystemen verursachte Last, z.B. durch automatisiertes Hinzufügen und Entfernen von Knoten innerhalb des DsN. Flexible Beschreibungssprachen wie [99] und Lösungen zur Ausbringung von Virtuellen Netzen wie [113] könnten mit den in dieser Arbeit vorgestellten Lösungen kombiniert werden, um Dienst-spezifische Netze auszubringen und effizient zu betreiben. Durch die Verbindungsmethoden und Bezugsmethoden des Zugriffsverfahrens oder das Hierarchische Knoten-Management ist eine Interaktion und ein Informationsaustausch zwischen Endsystemen und Dienst-spezifischen Netzen möglich, die eine dynamische Instantiierung eines Dienst-spezifischen Netzes oder eine Anpassung eines laufenden DsN ermöglichen. Darüber hinaus könnte für Dienstgüte, einen effizienteren Datenaustausch mit Dienst-spezifischen Netzen und mehr Flexibilität die Ausbringung von DsNs nahe der Endsysteme, z.B. in Netzen von Internet-Anbietern, untersucht werden. Dabei könnten für den Zugriff angepasste Infrastruktur-Protokolle verwendet werden, was durch die Verbindungsmethoden des Zugriffsverfahren ermöglicht wird. In diesem Zusammenhang ist auch eine Untersuchung von *Participatory Networking* [29] Ansätzen interessant, die mit *Software Defined Networking* Lösungen und einer Schnittstelle zu Anwendungen eine Kooperation zwischen Anwendungen und dem Netz erreichen. Mit einem solchen Ansatz könnte das Netz dem Endsystem geeignete Verbindungsmethoden und Zugangspunkte zu einem DsN basierend auf den Eigenschaften und Anforderungen des Dienstes sowie der aktuellen Netzlast mitteilen. Außerdem könnte in weiterführenden Arbeiten die Verwendung von DPDK-NENA als Plattform für die Ausbringung und den Betrieb Dienst-spezifischer Netze untersucht werden. Dabei wäre eine Integration von DPDK-NENA in *Software Defined Networking* Lösungen interessant.

Literatur

[1] *4WARD Project*. URL: http://www.4ward-project.eu/.

[2] V. Altmann u. a. "A DHT-Based Scalable Approach for Device and Service Discovery". In: *Embedded and Ubiquitous Computing (EUC), 2014 12th IEEE International Conference on*. 2014, S. 97–103. DOI: 10.1109/EUC.2014.23.

[3] Autonomic Network Architecture (ANA) Project. *http://www.ana-project.org*. 2010.

[4] M. Ayari u. a. "ADMA: autonomous decentralized management architecture for MANETs: a simple self-configuring case study". In: *IWCMC '09: Proceedings of the 2009 International Conference on Wireless Communications and Mobile Computing*. Leipzig, Germany: ACM, Juni 2009, S. 132–137. ISBN: 978-1-60558-569-7. DOI: http://doi.acm.org/10.1145/1582379.1582409.

[5] H. Backhaus, D. Martin und H. Wippel. *A Network Pluralist's Approach to Online Video Stores*. Demonstrator. Juli 2012.

[6] H. Backhaus u. a. *Tools for Application-tailored Network Engineering*. Talk. Juli 2012.

[7] S. Banerjee u. a. "Scalable Grid Service Discovery Based on UDDI". In: *Proceedings of the 3rd International Workshop on Middleware for Grid Computing*. MGC '05. Grenoble, France: ACM, 2005, S. 1–6. ISBN: 1-59593-269-0. DOI: 10.1145/1101499.1101501. URL: http://doi.acm.org/10.1145/1101499.1101501.

[8] A. Bavier u. a. "In VINI Veritas: Realistic and Controlled Network Experimentation". In: *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. SIGCOMM '06. Pisa, Italy: ACM, 2006, S. 3–14. ISBN: 1-59593-308-5. DOI: http://doi.acm.org/10.1145/1159913.1159916. URL: http://doi.acm.org/10.1145/1159913.1159916.

[9] T. Benson, A. Akella und D. A. Maltz. "Network Traffic Characteristics of Data Centers in the Wild". In: *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*. IMC '10. Melbourne, Australia: ACM, 2010, S. 267–280. ISBN: 978-1-4503-0483-2. DOI: 10.1145/1879141.1879175. URL: http://doi.acm.org/10.1145/1879141.1879175.

[10] T. Benson u. a. "Understanding Data Center Traffic Characteristics". In: *SIGCOMM Comput. Commun. Rev.* 40.1 (Jan. 2010), S. 92–99. ISSN: 0146-4833. DOI: 10.1145/1672308.1672325. URL: http://doi.acm.org/10.1145/1672308.1672325.

[11] Berners-Lee u. a. *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986. Jan. 2005.

[12] S. Bhatia u. a. "Trellis: A Platform for Building Flexible, Fast Virtual Networks on Commodity Hardware". In: *Proceedings of the 2008 ACM CoNEXT Conference*. CoNEXT '08. Madrid, Spain: ACM, 2008, 72:1–72:6. ISBN: 978-1-60558-210-8. DOI: http://doi.acm.org/10.1145/1544012.1544084. URL: http://doi.acm.org/10.1145/1544012.1544084.

[13] R. Bless u. a. "SpoVNet: An Architecture for Easy Creation and Deployment of Service Overlays". In: *Future Internet Services and Service Architectures*. Hrsg. von A. R. Prasad, J. F. Buford und V. K. Gurbani. River Publishers, Juni 2011, S. 23–47. ISBN: 87-92329-59-4.

[14] R. Braden u. a. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205 (Proposed Standard). Updated by RFCs 2750, 3936, 4495, 5946, 6437, 6780. Internet Engineering Task Force, Sep. 1997. URL: http://www.ietf.org/rfc/rfc2205.txt.

[15] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159 (Proposed Standard). Internet Engineering Task Force, März 2014. URL: http://www.ietf.org/rfc/rfc7159.txt.

[16] T. Bray u. a. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C Recommendation. http://www.w3.org/TR/2008/REC-xml-20081126/. W3C, Nov. 2008.

[17] S. Cheshire und M. Krochmal. *DNS-Based Service Discovery*. RFC 6763 (Proposed Standard). Internet Engineering Task Force, Feb. 2013. URL: http://www.ietf.org/rfc/rfc6763.txt.

[18] S. Cheshire und M. Krochmal. *Multicast DNS*. RFC 6762 (Proposed Standard). Internet Engineering Task Force, Feb. 2013. URL: http://www.ietf.org/rfc/rfc6762.txt.

[19] N. M. K. Chowdhury und R. Boutaba. "A Survey of Network Virtualization". In: *Comput. Netw.* 54.5 (Apr. 2010), S. 862–876. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2009.10.017. URL: http://dx.doi.org/10.1016/j.comnet.2009.10.017.

[20] B. Cohen. *The BitTorrent Protocol Specification*. http://www.bittorrent.org/beps/bep_0003.html. Jan. 2008.

[21] S. Deering und R. Hinden. *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard). Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112. Internet Engineering Task Force, Dez. 1998. URL: http://www.ietf.org/rfc/rfc2460.txt.

[22] L. Deri. “Improving Passive Packet Capture: Beyond Device Polling”. In: *Proceedings of SANE 2004*.

[23] R. Dingledine, N. Mathewson und P. Syverson. “Tor: The Second-generation Onion Router”. In: *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*. SSYM’04. San Diego, CA: USENIX Association, 2004, S. 21–21. URL: http://dl.acm.org/citation.cfm?id=1251375.1251396.

[24] R. Droms. *Dynamic Host Configuration Protocol*. RFC 2131 (Draft Standard). Updated by RFCs 3396, 4361, 5494, 6842. Internet Engineering Task Force, März 1997. URL: http://www.ietf.org/rfc/rfc2131.txt.

[25] R. Dutta u. a. “The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet”. In: *Proc. of IEEE International Conference on Communications (ICC)*. Glasgow, Scotland, UK, Juni 2007, S. 1899–1904. DOI: http://dx.doi.org/10.1109/ICC.2007.316.

[26] P. Emmerich u. a. “Assessing Soft-and Hardware Bottlenecks in PC-based Packet Forwarding Systems”. In: *ICN 2015* (2015), S. 90.

[27] D. Farinacci u. a. *Generic Routing Encapsulation (GRE)*. RFC 2784 (Proposed Standard). Updated by RFC 2890. Internet Engineering Task Force, März 2000. URL: http://www.ietf.org/rfc/rfc2784.txt.

[28] N. Feamster, L. Gao und J. Rexford. “How to Lease the Internet in Your Spare Time”. In: *SIGCOMM Comput. Commun. Rev.* 37 (1 2007), S. 61–64. ISSN: 0146-4833. DOI: http://doi.acm.org/10.1145/1198255.1198265. URL: http://doi.acm.org/10.1145/1198255.1198265.

[29] A. D. Ferguson u. a. “Participatory Networking: An API for Application Control of SDNs”. In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM ’13. Hong Kong, China: ACM, 2013, S. 327–338. ISBN: 978-1-4503-2056-6. DOI: 10.1145/2486001.2486003. URL: http://doi.acm.org/10.1145/2486001.2486003.

[30] R. Fielding u. a. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585. Internet Engineering Task Force, Juni 1999. URL: http://www.ietf.org/rfc/rfc2616.txt.

[31] M. Franzke u. a. *In-Network Management Concept*. Techn. Ber. 4WARD Project, 2009.

[32] G-Lab. *German Lab Project*. Online: http://www.german-lab.de. Letzter Zugriff: April 2015.

[33] T. Gamer und H. Wippel. “A Collaborative Attack Detection and its Challenges in the Future Internet”. In: *Proc. of the Joint ITG, ITC, and Euro-NF Workshop "Visions of Future Generation Networks"(EuroView)*. Aug. 2010.

[34] T. Gamer, C. P. Mayer und M. Zitterbart. “Distack—A Framework for Anomaly-based Large-scale Attack Detection”. In: *Proc. of 2nd International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*. Cap Esterel, France, Aug. 2008, S. 34–40.

[35] A. G. Ganek und T. A. Corbi. “The Dawning of the Autonomic Computing Era”. In: *IBM Syst. J.* 42.1 (Jan. 2003), S. 5–18. ISSN: 0018-8670. DOI: 10.1147/sj.421.0005. URL: http://dx.doi.org/10.1147/sj.421.0005.

[36] A. Gonzalez u. a. *In-network management design*. Project Deliverable D4.3. 4WARD Consortium, Jan. 2010.

[37] E. Guttman u. a. *Service Location Protocol, Version 2*. RFC 2608 (Proposed Standard). Updated by RFC 3224. Internet Engineering Task Force, Juni 1999. URL: http://www.ietf.org/rfc/rfc2608.txt.

[38] S. Han u. a. “PacketShader: A GPU-accelerated Software Router”. In: *Proceedings of the ACM SIGCOMM 2010 Conference*. SIGCOMM ’10. New Delhi, India: ACM, 2010, S. 195–206. ISBN: 978-1-4503-0201-2. DOI: 10.1145/1851182.1851207. URL: http://doi.acm.org/10.1145/1851182.1851207.

[39] R. Hancock u. a. *Next Steps in Signaling (NSIS): Framework*. RFC 4080 (Informational). Internet Engineering Task Force, Juni 2005. URL: http://www.ietf.org/rfc/rfc4080.txt.

[40] O. Hanka und H. Wippel. “Secure Deployment of Application-tailored Protocols in Future Networks”. In: *Network of the Future (NOF), 2011 International Conference on the*. 2011, S. 10–14. DOI: 10.1109/NOF.2011.6126668.

[41] O. Hanka u. a. “A Distributed Public Key Infrastructure based on Threshold Cryptography for the HiiMap Next Generation Internet Architecture”. In: *Future Internet* 3.1 (2011), S. 14–30. ISSN: 1999-5903.

[42] O. Hanka u. a. “HiiMap: Hierarchical Internet Mapping Architecture”. In: *In First International Conference on Future Information Networks, Beijing, China, P.R. China* (2009).

[43] D. Harrington, R. Presuhn und B. Wijnen. *An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*. RFC 3411. The Internet Engineering Task Force (IETF), Dez. 2002. URL: http://tools.ietf.org/html/rfc3411.

[44] R. Housley und S. Hollenbeck. *EtherIP: Tunneling Ethernet Frames in IP Datagrams*. RFC 3378 (Informational). Internet Engineering Task Force, Sep. 2002. URL: http://www.ietf.org/rfc/rfc3378.txt.

[45] J. Hwang, K. K. Ramakrishnan und T. Wood. "NetVM: High Performance and Flexible Networking Using Virtualization on Commodity Platforms". In: *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*. Seattle, WA: USENIX Association, Apr. 2014, S. 445–458. ISBN: 978-1-931971-09-6. URL: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/hwang.

[46] "IEEE Standard for Ethernet - Section 1". In: *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)* (2012), S. 1–580. DOI: 10.1109/IEEESTD.2012.6419735.

[47] "IEEE Standard for Information Technology- Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 3: Carrier Sense Multiple Access With Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Media Access Control (MAC) Parameters, Physical Layers, and Management Parameters for 10 Gb/S Operation". In: *IEEE Std 802.3ae-2002 (Amendment to IEEE Std 802.3-2002)* (2002), S. 1–516. DOI: 10.1109/IEEESTD.2002.94131.

[48] "IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks". In: *IEEE Std 802.1Q-2011 (Revision of IEEE Std 802.1Q-2005)* (2011), S. 1–1365. DOI: 10.1109/IEEESTD.2011.6009146.

[49] Intel. *DPDK: Data Plane Development Kit*. Online: http://dpdk.org. Letzter Zugriff: Nov. 2014.

[50] Intel. *DPDK Getting Started Guide for Linux v. 1.7.1*. Online: http://dpdk.org/doc/guides/linux_gsg/. Letzter Zugriff: Dez. 2014.

[51] Intel. *DPDK Programmer's Guide v. 1.7.1*. Online: http://dpdk.org/doc/guides/prog_guide/. Letzter Zugriff: Dez. 2014.

[52] Intel. *DPDK Sample Applications User Guide v. 1.7.1*. Online: http://dpdk.org/doc/guides/sample_app_ug/. Letzter Zugriff: Dez. 2014.

[53] Intel Open Source 01.org. *Packet Processing - Intel DPDK vSwitch*. Online: https://01.org/packet-processing. Letzter Zugriff: Dez. 2014.

[54] ITU-T editor. *Principles for a telecommunications management network*. ITU-T Recommendation M.3010. International Telecommunication Union - Telecommunication Standardization Sector (ITU-T), Feb. 2000. URL: http://www.itu.int/rec/T-REC-M.3010/en.

[55] V. Jacobson u. a. “Networking Named Content”. In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT ’09. Rome, Italy: ACM, 2009, S. 1–12. ISBN: 978-1-60558-636-6. DOI: 10.1145/1658939.1658941. URL: http://doi.acm.org/10.1145/1658939.1658941.

[56] W. John und S. Tafvelin. “Analysis of Internet Backbone Traffic and Header Anomalies Observed”. In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. IMC ’07. San Diego, California, USA: ACM, 2007, S. 111–116. ISBN: 978-1-59593-908-1. DOI: 10.1145/1298306.1298321. URL: http://doi.acm.org/10.1145/1298306.1298321.

[57] A. Keller u. a. “A System Architecture for Evolving Protocol Stacks”. In: *Proceedings of 17th International Conference on Computer Communications and Networks (ICCCN)*. St. Thomas, US Virgin Islands, Aug. 2008. DOI: 10.1109/ICCCN.2008.ECP.44.

[58] S. Kent und K. Seo. *Security Architecture for the Internet Protocol*. RFC 4301 (Proposed Standard). Updated by RFC 6040. Internet Engineering Task Force, Dez. 2005. URL: http://www.ietf.org/rfc/rfc4301.txt.

[59] J. O. Kephart und D. M. Chess. “The Vision of Autonomic Computing”. In: *Computer* 36.1 (Jan. 2003), S. 41–50. ISSN: 0018-9162. DOI: 10.1109/MC.2003.1160055. URL: http://dx.doi.org/10.1109/MC.2003.1160055.

[60] E. Kohler, M. Handley und S. Floyd. *Datagram Congestion Control Protocol (DCCP)*. RFC 4340 (Proposed Standard). Updated by RFCs 5595, 5596, 6335, 6773. Internet Engineering Task Force, März 2006. URL: http://www.ietf.org/rfc/rfc4340.txt.

[61] M. Kounavis u. a. “The Genesis Kernel: A Programming System for Spawning Network Architectures”. In: *Selected Areas in Communications, IEEE Journal on* 19.3 (2001), S. 511 –526. ISSN: 0733-8716. DOI: 10.1109/49.917711.

[62] A. Kousaridas u. a. “Future internet elements: cognition and self-management design issues”. In: *Autonomics ’08: Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems*. Turin, Italy: ICST (Institute for Computer Sciences, Social-Informatics und Telecommunications Engineering), Sep. 2008, S. 1–6. ISBN: 978-963-9799-34-9.

[63] T. Lang, P. Branch und G. Armitage. “A Synthetic Traffic Model for Quake3”. In: *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*. ACE ’04. Singapore: ACM, 2004, S. 233–238. ISBN: 1-58113-882-2. DOI: 10.1145/1067343.1067373. URL: http://doi.acm.org/10.1145/1067343.1067373.

[64] J. W. Lockwood u. a. “Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)”. In: *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*. FPGA ’01. Monterey, California, USA: ACM, 2001, S. 87–93. ISBN: 1-58113-341-3. DOI: 10.1145/360276.360304. URL: http://doi.acm.org/10.1145/360276.360304.

[65] J. Lockwood u. a. “NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing”. In: *Microelectronic Systems Education, 2007. MSE ’07. IEEE International Conference on*. 2007, S. 160–161. DOI: 10.1109/MSE.2007.69.

[66] M. Mahalingam u. a. *Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks*. RFC 7348 (Informational). Internet Engineering Task Force, Aug. 2014. URL: http://www.ietf.org/rfc/rfc7348.txt.

[67] I. Marinos, R. N. M. Watson und M. Handley. “Network Stack Specialization for Performance”. In: *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. HotNets-XII. College Park, Maryland: ACM, 2013, 9:1–9:7. ISBN: 978-1-4503-2596-7. DOI: 10.1145/2535771.2535779. URL: http://doi.acm.org/10.1145/2535771.2535779.

[68] D. Martin und H. Wippel. *API Usage and Message Passing in NENA*. Techn. Ber. TM-2013-1. Institute of Telematics, Karlsruhe Institute of Technology, Jan. 2013.

[69] D. Martin und H. Wippel. “Evaluating a framework for different networking paradigms”. In: *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*. 2013, S. 288–291. DOI: 10.1109/LCN.2013.6761251.

[70] D. Martin und H. Wippel. *Evaluating a Framework for Different Networking Paradigms (TR)*. Techn. Ber. TM-2013-2. Institute of Telematics, Karlsruhe Institute of Technology, Juli 2013.

[71] D. Martin, H. Wippel und H. Backhaus. “A Future-Proof Application-to-Network Interface”. In: *Proceedings of the 2011 Second International Conference on the Network of the Future (NoF 2011)*. IEEE, Nov. 2011.

[72] D. Martin u. a. *Designing and Running Concurrent Future Networks (Poster+Demo)*. Demonstrator. Okt. 2009.

[73] D. Martin. *NENA - Ein Rahmenwerk für Maßgeschneiderte Kommunikationsnetze*. Verlag Dr. Hut, Juni 2014. ISBN: 978-3-8439-1657-8.

[74] D. Martin, L. Völker und M. Zitterbart. “A Flexible Framework for Future Internet Design, Assessment, and Operation”. In: *Computer Networks* 55.4 (Feb. 2011), S. 910–918. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2010.12.015.

[75] M. May u. a. “Monitoring as first class citizen in an autonomic network universe”. In: *Bionetics 2007. 2nd International Conference on Bio-Inspired Models of Network, Information and Computing Systems*. Budapest, Hungary, Dez. 2007, S. 247–254. DOI: 10.1109/BIMNICS.2007.4610121.

[76] P. Maymounkov und D. Mazières. “Kademlia: A Peer-to-Peer Information System Based on the XOR Metric”. English. In: *Peer-to-Peer Systems*. Hrsg. von P. Druschel, F. Kaashoek und A. Rowstron. Bd. 2429. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, S. 53–65. ISBN: 978-3-540-44179-3. DOI: 10.1007/3-540-45748-8_5. URL: http://dx.doi.org/10.1007/3-540-45748-8_5.

[77] J. van der Merwe u. a. “The Tempest-A Practical Framework for Network Programmability”. In: *Network, IEEE* 12.3 (1998), S. 20 –28. ISSN: 0890-8044. DOI: 10.1109/65.690958.

[78] S. Mies, O. Waldhorst und H. Wippel. “Towards End-to-End Connectivity for Overlays Across Heterogeneous Networks”. In: *Communications Workshops, 2009. ICC Workshops 2009. IEEE International Conference on*. 2009, S. 1–6. DOI: 10.1109/ICCW.2009.5207975.

[79] M. Mitchell, J. Oldham und A. Samuel. *Advanced Linux Programming*. Thousand Oaks, CA, USA: New Riders Publishing, 2001. ISBN: 0735710430.

[80] P. Mockapetris. *Domain names - concepts and facilities*. RFC 1034 (INTERNET STANDARD). Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936. Internet Engineering Task Force, Nov. 1987. URL: http://www.ietf.org/rfc/rfc1034.txt.

[81] L. R. Monnerat und C. L. Amorim. “D1HT: A Distributed One Hop Hash Table”. In: *Proceedings 20th IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2006.

[82] D. Murray und T. Koziniec. “The state of enterprise network traffic in 2012”. In: *Communications (APCC), 2012 18th Asia-Pacific Conference on*. 2012, S. 179–184. DOI: 10.1109/APCC.2012.6388126.

[83] ntop. *PF_RING User Guide v. 6.0.0*. Online: https://svn.ntop.org/svn/ntop/trunk/PF_RING/doc/UsersGuide.pdf. Letzter Zugriff: Dez. 2014.

[84] OpenVPN Technologies, Inc. *OpenVPN - Open Source VPN*. Online: http://openvpn.net. Letzter Zugriff: April 2015.

[85] OvS. *Open vSwitch*. Online: http://openvswitch.org. Letzter Zugriff: Dez. 2014.

[86] S. Perez Sanchez und R. Bless. “Network Design”. In: *Architecture and Design for the Future Internet*. Signals and Communication Technology. Springer Netherlands, 2011, S. 59–87. ISBN: 978-90-481-9346-2. URL: http://dx.doi.org/10.1007/978-90-481-9346-2_4.

[87] C. Perkins. *IP Encapsulation within IP*. RFC 2003 (Proposed Standard). Updated by RFCs 3168, 6864. Internet Engineering Task Force, Okt. 1996. URL: http://www.ietf.org/rfc/rfc2003.txt.

[88] J. Postel. *Internet Protocol*. RFC 791 (INTERNET STANDARD). Updated by RFCs 1349, 2474, 6864. Internet Engineering Task Force, Sep. 1981. URL: http://www.ietf.org/rfc/rfc791.txt.

[89] J. Postel. *Transmission Control Protocol*. RFC 793 (INTERNET STANDARD). Updated by RFCs 1122, 3168, 6093, 6528. Internet Engineering Task Force, Sep. 1981. URL: http://www.ietf.org/rfc/rfc793.txt.

[90] J. Postel. *User Datagram Protocol*. RFC 768 (INTERNET STANDARD). Internet Engineering Task Force, Aug. 1980. URL: http://www.ietf.org/rfc/rfc768.txt.

[91] J. Postel und J. Reynolds. *File Transfer Protocol*. RFC 959 (INTERNET STANDARD). Updated by RFCs 2228, 2640, 2773, 3659, 5797, 7151. Internet Engineering Task Force, Okt. 1985. URL: http://www.ietf.org/rfc/rfc959.txt.

[92] D. Raychaudhuri u. a. “CogNet: an architectural foundation for experimental cognitive radio networks within the future internet”. In: *MobiArch ’06: Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture*. San Francisco, California: ACM, Dez. 2006, S. 11–16. ISBN: 1-59593-566-5. DOI: http://doi.acm.org/10.1145/1186699.1186707.

[93] R. Rivest. *The MD5 Message-Digest Algorithm*. RFC 1321 (Informational). Updated by RFC 6151. Internet Engineering Task Force, Apr. 1992. URL: http://www.ietf.org/rfc/rfc1321.txt.

[94] L. Rizzo. “Netmap: A Novel Framework for Fast Packet I/O”. In: *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*. USENIX ATC’12. Boston, MA: USENIX Association, 2012. URL: http://dl.acm.org/citation.cfm?id=2342821.2342830.

[95] L. Rizzo und G. Lettieri. “VALE, a Switched Ethernet for Virtual Machines”. In: *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*. CoNEXT ’12. Nice, France: ACM, 2012, S. 61–72. ISBN: 978-1-4503-1775-7. DOI: 10.1145/2413176.2413185. URL: http://doi.acm.org/10.1145/2413176.2413185.

[96] M. Röhricht, H. Wippel und M. Zitterbart. *Towards Composed Communication Services in Future Networks*. Talk. Juli 2009.

[97] R. Roth u. a. *In-Network Management: Definition of scenarios and use cases*. Techn. Ber. 4WARD Project, Apr. 2009.

[98] T. Sader. "Entwurf einer Anwendungssschnittstelle für ein DPDK-basiertes NENA-Rahmenwerk". Betreuer: Hans Wippel. Studienarbeit. Karlsruhe Institute of Technology (KIT), März 2015.

[99] G. Schaffrath u. a. "A Resource Description Language with Vagueness Support for Multi-Provider Cloud Networks". In: *Proc. International Conference on Computer Communication Networks (ICCCN)*. 2012.

[100] S. Schuetz u. a. "Autonomic and decentralized management of wireless access networks". In: *Network and Service Management, IEEE Transactions on* 4.2 (2007), S. 96–106. ISSN: 1932-4537. DOI: 10.1109/TNSM.2007.070905.

[101] *FIRE: Future Internet Research and Experimentation*. Techn. Ber. FIRE initiative, Aug. 2009.

[102] A. Shamir. "How to share a Secret". In: *Communications of the ACM* 22.11 (1979), S. 612–613.

[103] S. Sharafeddine u. a. "On traffic characteristics and bandwidth requirements of voice over IP applications". In: *Computers and Communication, 2003. (ISCC 2003). Proceedings. Eighth IEEE International Symposium on*. 2003, 1324–1330 vol.2. DOI: 10.1109/ISCC.2003.1214297.

[104] R. Sinha, C. Papadopoulos und J. Heidemann. *Internet Packet Size Distributions: Some Observations*. Techn. Ber. ISI-TR-2007-643. Orignally released October 2005 as web page http://netweb.usc.edu/~rsinha/pkt-sizes/. USC/Information Sciences Institute, 2007. URL: http://www.isi.edu/~johnh/PAPERS/Sinha07a.html.

[105] M. Sridharan u. a. *NVGRE: Network Virtualization using Generic Routing Encapsulation*. Internet Draft. Internet Engineering Task Force, 2013. URL: http://tools.ietf.org/id/draft-sridharan-virtualization-nvgre-03.txt.

[106] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335, 7053. Internet Engineering Task Force, Sep. 2007. URL: http://www.ietf.org/rfc/rfc4960.txt.

[107] I. Stoica u. a. "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications". In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '01. San Diego, California, USA: ACM, 2001, S. 149–160. ISBN: 1-58113-411-8. DOI: 10.1145/383059.383071. URL: http://doi.acm.org/10.1145/383059.383071.

[108] *The SelfNET Project Homepage*. https://www.ict-selfnet.eu/. 2010.

[109] The Tinc development team. *tinc*. Online: http://www.tinc-vpn.org. Letzter Zugriff: April 2015.

[110] W. Townsley u. a. *Layer Two Tunneling Protocol L2TP*. RFC 2661 (Proposed Standard). Internet Engineering Task Force, Aug. 1999. URL: http://www.ietf.org/rfc/rfc2661.txt.

[111] J. Ubillos u. a. *Name-Based Sockets Architecture*. Internet Draft (draft-ubillos-name-based-sockets-03). März 2010.

[112] *UDDI Version 3.0.2. UDDI Spec Technical Committee Draft*. OASIS Standard 1 July 2009. Organization for the Advancement of Structured Information Standards, Okt. 2004. URL: http://www.uddi.org/pubs/uddi_v3.htm.

[113] I. Vaishnavi u. a. "An Architecture for Creating and Managing Virtual Networks". In: *Proc. 24th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*. 2013.

[114] M. Vogt u. a. *Advanced Packaging Tool*. 2011. URL: http://packages.qa.debian.org/a/apt.html.

[115] L. Völker u. a. "Design Process and Development Tools for Concurrent Future Networks". In: *3rd GI/ITG KuVS Workshop on The Future Internet*. Mai 2009.

[116] *Web Services Dynamic Discovery (WS-Discovery) Version 1.1*. OASIS Standard 1 July 2009. Organization for the Advancement of Structured Information Standards, Juli 2009. URL: http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.html.

[117] P. Willmann, S. Rixner und A. L. Cox. "An Evaluation of Network Stack Parallelization Strategies in Modern Operating Systems." In: *USENIX Annual Technical Conference, General Track*. 2006, S. 91–96.

[118] H. Wippel, T. Gamer und D. Martin. "A Hierarchical Node Management System for Application-tailored Network Protocols and Architectures". In: *5th GI/ITG KuVS Workshop on Future Internet*. Gesellschaft für Informatik, Juni 2010.

[119] H. Wippel und O. Hanka. "End User Node Access to Application-Tailored Future Networks". In: *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. 2012, S. 1–7. DOI: 10.1109/ICCCN.2012.6289290.

[120] H. Wippel u. a. "Hierarchical Node Management in the Future Internet". In: *Communications Workshops (ICC), 2011 IEEE International Conference on*. 2011, S. 1–5. DOI: 10.1109/iccw.2011.5963590.

[121] H. Wippel. “DPDK-based Implementation of Application-tailored Networks on End User Nodes”. In: *Network of the Future (NOF), 2014 International Conference on the*. 2014.

[122] H. Wippel und S. Finster. “Smart Metering Using Application-tailored Networks”. In: *Proceedings of the Fourth International Conference on Future Energy Systems*. e-Energy ’13. Berkeley, California, USA: ACM, 2013, S. 293–294. ISBN: 978-1-4503-2052-8. DOI: 10.1145/2487166.2487213. URL: http://doi.acm.org/10.1145/2487166.2487213.

[123] H. Wippel und O. Hanka. “Deployment of Application-Tailored Protocols in Future Networks”. In: *Proceedings of the 11th Euroview Future Internet Workshop*. Würzburg, Germany, 2011.

[124] H. Wippel u. a. “Evaluation of Future Network Architectures and Services in the G-Lab Testbed”. English. In: *Testbeds and Research Infrastructures. Development of Networks and Communities*. Hrsg. von T. Magedanz u. a. Bd. 46. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, 2011, S. 587–589. ISBN: 978-3-642-17850-4. DOI: 10.1007/978-3-642-17851-1_49. URL: http://dx.doi.org/10.1007/978-3-642-17851-1_49.

[125] F. Wuhib, M. Dam und R. Stadler. “A gossiping protocol for detecting global threshold crossings”. In: *Network and Service Management, IEEE Transactions on* 7.1 (2010), S. 42–57. ISSN: 1932-4537. DOI: 10.1109/TNSM.2010.I9P0329.

[126] F. Wuhib u. a. “Robust Monitoring of Network-wide Aggregates through Gossiping”. In: *Integrated Network Management, 2007. IM ’07. 10th IFIP/IEEE International Symposium on*. 2007, S. 226–235. DOI: 10.1109/INM.2007.374787.

[127] F. Wuhib, M. Dam und R. Stadler. “Decentralized Detection of Global Threshold Crossings Using Aggregation Trees”. In: *Comput. Netw.* 52.9 (Juni 2008), S. 1745–1761. ISSN: 1389-1286. DOI: 10.1016/j.comnet.2008.02.015. URL: http://dx.doi.org/10.1016/j.comnet.2008.02.015.

[128] K.-K. Yap u. a. “Towards Software-friendly Networks”. In: *Proceedings of the First ACM Asia-pacific Workshop on Systems*. APSys ’10. New Delhi, India: ACM, 2010, S. 49–54. ISBN: 978-1-4503-0195-4. DOI: 10.1145/1851276.1851288. URL: http://doi.acm.org/10.1145/1851276.1851288.

[129] T. Ylonen und C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251 (Proposed Standard). Internet Engineering Task Force, Jan. 2006. URL: http://www.ietf.org/rfc/rfc4251.txt.

[130] S. Zander und G. Armitage. “A Traffic Model for the Xbox Game Halo 2”. In: *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*. NOSSDAV ’05. Stevenson, Washington, USA: ACM, 2005, S. 13–18. ISBN: 1-58113-987-X. DOI: 10.1145/1065983.1065987. URL: http://doi.acm.org/10.1145/1065983.1065987.

[131] D. Zhou u. a. “Scalable, High Performance Ethernet Forwarding with CuckooSwitch”. In: *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. CoNEXT ’13. Santa Barbara, California, USA: ACM, 2013, S. 97–108. ISBN: 978-1-4503-2101-3. DOI: 10.1145/2535372.2535379. URL: http://doi.acm.org/10.1145/2535372.2535379.